Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Federal Department of the Environment, Transport, Energy and Communications DETEC

Swiss Federal Office of Energy SFOE
Energy Research and Cleantech Division

Final report dated December 15, 2023

Renewable Management and Real-Time Control Platform (ReMaP)

Photo: The NEST building at Empa. The NEST building and the move platform are accessible through ReMaP. © Zooey Braun.

Date: December 15, 2023

Location: Bern

Subsidiser:
Swiss Federal Office of Energy SFOE
Energy Research and Cleantech Section
CH-3003 Bern
www.bfe.admin.ch

Co-Financing:
ETH Zürich Foundation
Weinbergstrasse 29, 8006 Zürich
www.ethz-foundation.ch

ETH Zürich
Rämistrasse 101, 8092 Zürich
www.ethz.ch

Empa
Überlandstrasse 129, 8600 Dübendorf
www.empa.ch

Paul Scherrer Institut
Forschungsstrasse 111, 5232 Villigen
www.psi.ch

EVUlution AG (formerly smart grid solutions)
Bahnhofstrasse 11, 7302 Landquart
https://www.evulution.com/

Secure Switzerland AG (former Adaptricity)
Buckhauserstrasse 1, 8048 Zurich
www.adaptricity.com

National Instruments Schweiz GmbH
Mellingerstrasse 1, 5400 Baden
www.ni.com

Subsidy recipients:
ETH Zürich, Energy Science Center
Sonneggstrasse 28, 8006 Zurich
www.esc.ethz.ch

Empa
Überlandstrasse 129, 8600 Dübendorf
www.empa.ch

Paul Scherrer Institut (PSI)
Forschungsstrasse 111, 5232 Villigen
www.psi.ch

EVUlution AG (formerly smart grid solutions)
Bahnhofstrasse 11, 7302 Landquart
https://www.evulution.com/

Secure Switzerland AG (former Adaptricity)
Buckhauserstrasse 1, 8048 Zurich
www.adaptricity.com

Authors (in alphabetical order):
Prof. Konstantinos Boulouchos, ETH, boulouchos@lav.mavt.ethz.ch
Alain Brenzikofer, Super Computing Systems, alain.brenzikofer@scs.ch
Philippe Buchecker, ETH, bucheckp@fen.ethz.ch
Dr. Turhan Demiray, ETH, demirayt@fen.ethz.ch
Prof. Florian Dörfler, ETH, floriand@ethz.ch
Marta Fochesato, ETH, mfochesato@ethz.ch
Nicolas Früh, Adaptricity, nfrüh@adaptricity.com
Dr. Gianfranco Guidati, ETH, gianfranco.guidati@esc.ethz.ch
Philipp Heer, Empa, philipp.heer@empa.ch
Marcel Hofer, PSI, marcel.hofer@psi.ch
Shan-Shan Hsieh, ETH, hsieh@arch.ethz.ch
Prof. Gabriela Hug, ETH, hug@eeh.ee.ethz.ch
Prof. Maria Lukatskaya, ETH, mlukatskaya@ethz.ch
Roland Lüthy, smart grid solutions, roland.luethy@smartgridsolutions.ch
Prof. John Lygeros, ETH, jlygeros@ethz.ch
Dr. Adamantios Marinakis, ETH, marinakis@fen.ethz.ch
Miguel Picallo Cruz, ETH, miguelp@control.ee.ethz.ch
Sabine Proll, Super Computing Systems, sabine.proll@scs.ch
Prof. Giovanni Sansavini, ETH, sansavig@ethz.ch
Dr. Christian Schaffner, ETH, schaffner@esc.ethz.ch
Dr. Tilman Schildhauer, PSI, tilman.schildhauer@psi.ch
Prof. Arno Schlüter, ETH, schlueter@arch.ethz.ch
Prof. Thomas Justus Schmidt, PSI, thomasjustus.schmidt@psi.ch
Christian Schürch, ETH, schuercc@lav.mavt.ethz.ch
Nicolas Stocker, Adaptricity, nstocker@adaptricity.com
Dr. Sigita Trabesinger, PSI, sigita.trabesinger@psi.ch
Dr. Andreas Ulbig, Adaptricity, aulbig@adaptricity.com
Raphael Wu, ETH, wur@ethz.ch
Thierry Zufferey, ETH, thierryz@eeh.ee.ethz.ch

SFOE project coordinators:
Dr. Michael Moser, michael.moser@bfe.admin.ch
Dr. Karin Söderström, karin.soederstroem@bfe.admin.ch

SFOE contract number: SI/501810-1

Summary

Switzerland has set the goal to reach net-zero greenhouse gas emissions by 2050. Many teams have used energy system models to describe feasible pathways and this led to a consensus on many crucial aspects. The most important ones are the electrification of the mobility and heating sector via battery electric vehicles and heat pumps, respectively, and the dominant role of photovoltaics in the future energy mix. The challenges that come with this transformation from a fossil fuel system to an electrical system that relies on volatile generation are obvious. To address these challenges and to understand the implications of the widespread adoption of fluctuating renewable energy sources on the energy system as a whole and especially on the distribution grid is therefore of critical importance.

The “Renewable Management and Real-Time Control Platform” (ReMaP) project contributed to this understanding by developing a flexible, hardware- and software-based research platform for assessing potential energy-system solutions for the neighborhood of the future. This platform allows multi-energy systems on the distribution level to be tested, analyzed, and optimized and thereby encourage collaboration between academia and industry. A distinguishing feature of the platform is that it integrates the existing Energy System Integration platform at the Paul Scherrer Institut and the NEST, move, and ehub platforms at Empa.

The ReMaP project was based on two pillars: the first is the ReMaP platform itself that enables the aforementioned testing of future multi-energy systems – from a pure computer simulation, i.e. the ReSIM software, to a hardware-in-the-loop (HIL) approach where some or all elements are represented by real hardware. The second are a total of nine subprojects that focus on specific aspects of the challenges to build and control a future energy system. Most of these subprojects used elements of the ReMaP platform, many enriched the platform, e.g. by supplying novel control approaches in the form of software modules, some performed full HIL experiments involving ESI, Nest and even additional elements, such as a combined heat and power plant at ETH or a biogas plant in Inwil. This report serves as a guidance to the results that were obtained during the project.

The Subprojects T3.1-10 covered a number overlapping themes that may be grouped as: (i) innovative control and design approaches (T3.1, T3.3), (ii) dealing with uncertainty of loads (T3.2, T3.4), (iii) battery and fuel cell electric vehicles (T3.9, T3.10), and (iv) improved components (T3.5, T3.6, T3.7).

Innovative control and design approaches: The introduction of renewable energy sources and flexible demand, especially in distribution grids, opens new paths toward more sustainable power systems. Yet, the volatility and unpredictability of these elements, e.g., household demand or available renewable power, pose severe challenges on the reliability of the grid. Therefore, a faster and more efficient grid operation is essential for creating a more sustainable power system. **Subproject T3.1** explored different new control approaches based on Online Feedback-based Optimization (OFO). Simulations with standard test cases proved their effectiveness in cases with noisy and incomplete measurements, especially combining OFO with a dynamic estimation based on linear and also nonlinear power flow models. Also a model-free OFO was studied that learns the input-output sensitivity. This was achieved by combining a standard OFO controller with a recursive least-squares estimation to learn the sensitivity from measurements during real-time operation. Again, the performance of the approach was validated on a numerical simulation using the IEEE 123-bus test feeder, showing superior performance compared to a state-of-the-art OFO with a constant sensitivity approximation [1].

While the previously described results related mostly to distribution grid level control, the interaction of these low-voltage levels with the higher transmission grid levels is of paramount importance. **Subproject T3.3** was dedicated to this link focusing on Active electrical Distribution Networks (ADN) and the interaction with their hosting network. This was studied using the ReSIM simulation framework, including the Adaptricity grid simulation software, focusing on the impact of ADNs and electrification on voltage deviations, branch and transformer loading in a rural, a semi-urban and an urban hosting network. While the additional load due to electrification of heating and mobility can lead to voltage constraint violations in rural networks and to short-term branch overloads in semi-urban and urban networks, transitioning 50% of the load buses to optimally designed ADNs can solve voltage and overload issues in all investigated networks with two complementary mechanisms: (i) The distributed energy resources (DER)

within ADNs enable a higher self-consumption, which reduces the total load in the network. (ii) An energy management system was developed and tested in ReSIM, which uses the available DER capacity to mitigate the imbalance between load and generation caused by day-ahead forecast errors and to alleviate voltage and overload issues in the hosting networks [14].

Dealing with uncertainty of loads: For better grid operation, the use of short-term predictions, be it minutes or days in advance, at the customer level is of great help. The granular data collected via smart meters and appliance sensors can be used as input to machine learning approaches that then provide predictions for the desired time range. At the transmission level or highly aggregated level, deterministic forecasting algorithms can properly predict the load which is relatively smooth and periodic. However, the load in distribution grids, and especially at the customer level, is highly volatile such that deterministic approaches are not satisfactory. Hence, due to the inherent uncertainties and inaccuracies of predictions, the approach should provide probabilistic predictions, i.e., distributions and probability quantiles instead of deterministic values. **Subproject T3.2** tackled this problem by developing probabilistic forecasting algorithms that can indeed cover the entire uncertainty range without assuming specific error distributions and can update their prediction in real time. To achieve this, historical data was used to train multiple regression models, generating independent forecasts. These models were then combined into an ensemble model for enhanced accuracy. Both deterministic and probabilistic models were applied to both building and individual appliance levels. Results showed that deterministic models are less effective due to load volatility, while probabilistic models provided improved uncertainty intervals. This approach was integrated as a software package into ReSIM to predict building-level power flows [6].

Coupling the heating and electricity sector is a prerequisite to reach the Swiss climate goals. While heat pumps are obviously the technology of the future, their widespread use increases uncertainty of future electricity demand, on an annual level but also in shorter time scales. **Subproject T3.4** dealt with the aspect of buildings and their interaction with future energy systems [21]. Three district archetypes (urban, sub-urban and rural) were defined and simulated using the City Energy Analyst (CEA), an open-source computational framework for the analysis of urban energy systems. CEA uses dynamic and multi-physics models to forecast the load profile of heating, cooling, and electricity of single and multiple buildings up to entire districts, in hourly time-steps. In addition, a plug-in for building stock transformation was developed to generate input scenarios for energy demand simulation. The results showed that by 2060, in the urban district archetype, space cooling demand can reach equivalent amounts to space heating demand, leading to important considerations with regards to systems design and optimization in these areas – especially in connection with the electricity system. In the sub-urban district archetype, the findings revealed that the energy planning can prove to be very challenging, depending on the urban development trajectories as well as the energy efficiency strategies. Contrary, in the rural district archetype, these difficulties were found to be less pronounced, with space heating demand possibly dominating the decision-making process, yet much depending on the local regional climate.

Battery and fuel cell electric vehicles: **Subproject T3.9** aimed at developing and analyzing advanced control algorithms for the optimal operations of future energy systems, with a specific focus on the integration between the power and mobility sector. A modeling library with detailed models of the devices at the ESI platform and at the move platform was developed that can capture the nonlinearities of real-world hardware components and ultimately support the development of accurate model-based simulations and control schemes. These devices included electrolyzers, fuel cells, gas cleaners, compressors and storage tanks. The project also provided advanced control algorithms for the optimal control of future energy hubs that are capable to deal with the inherent stochasticity affecting such systems and come with certificates in terms of robustness, computational footprint and scalability. An Online Optimization Algorithm was developed that solves an optimization problem for the real-time control of complex systems. The main focus here was on the computational aspect: a dual dynamic programming framework was provided for the control of nonlinear problems over long horizon (for example, for seasonal energy storage) with an optimal trade-off between accuracy and computation. Another example was an Online Peak Shaving Optimization controller that allowed to shave the peak of an EV charging station equipped with an hydrogen energy storage system. This controller was tested using the ReMaP platform in a HIL experiment on the ESI platform of PSI. These tests confirmed that

the performance was superior to existing control schemes in limiting the peak power seen by the charging station [30].

A very similar problem of a fast charging station at a highway was studied with HIL experiments in **Subproject T3.10**. Also in this case, it could be shown how the combination with a hydrogen storage can help to minimize peak charging power. A second case considered the grid-neutral and renewable hydrogen production for a fuel cell vehicle fleet. Here, an electrolyzer was considered together with a Power-to-Gas seasonal storage system, which consists of a methanation, the gas grid as intermediate storage and a steam reformer. The dynamic operation of the plant over the year was simulated. The simulation study showed that in the next decades further decreasing costs for the electrolysis and the PV system as well as increased price for hydrogen make the grid-neutral renewable hydrogen production economically feasible. For the experimental campaign, the electrolysis, gas cleaning, and H2-tank at PSI, the container based methanation unit connected to a real biogas plant in Inwil as well as the battery at Empa, all connected via the ReMaP framework, were used for a HIL experiment. It could be shown that the methanation plant is able to handle part load changes without compromising the gas quality [39].

Improved components: The ReMaP project stressed the *system* aspect of future energy systems, nevertheless, a system is made up from components, three of which were studied in detail in Subprojects T3.5 (combined heat and power plants, CHP), T3.6 (batteries) and T3.7 (supercapacitors). CHP plants convert chemical energy into thermal and electrical energy. Such plants can be started up quickly and have great potential to contribute to a stable future energy system based largely on fluctuating renewable energy sources. Therefore, the overall goal of **Subproject T3.5** was to find a way to increase both exergy efficiency and flexibility of such plants [25]. Additionally, the ability to store energy seasonally was investigated. The focus was on micro CHP plants with an electrical output power of below 10 kW but the developed concepts should also be able to suit bigger plants to a certain extent. Two options to improve the flexibility and exergy efficiency of a conventional CHP system were investigated with simulations. The more promising option was further tested with hardware-in-the-loop (HIL) experiments on the ReMaP platform (see also Section 3.1). It was seen that a steam methane reformer (SMR) CHP concept is to be favored against the concept of combining the CHP with a ground thermal storage system and a heat pump. Such a system promises to be more flexible (increased degree of self-sufficiency by 14.0 percentage points) and exergy efficient (increase of 2.9 percentage points) than a conventional system.

Batteries will be a key component of future energy systems – in battery electric vehicles and as stationary electricity storage devices. The goal of **Subproject T3.6** was to gain insights into the factors that contribute to battery degradation and to improve the longevity of batteries targeting stationary applications [26]. To achieve this, a cycling aging campaign was conducted to simulate real-world conditions, including both standard and "abuse" scenarios. Four different temperatures, two different charge/discharge rates and three different Depth of Discharge (DOD) levels were tested. Various parameters were monitored during the aging process, where the main parameters of interest were the efficiency loss per cycle and the discharge capacity loss. Both are useful to predict the life of the battery cell. The main findings were that the temperature had a strong influence on the performance of all three cell technologies (NMC/LTO, NMC/Gr, and LFP/Gr) studied. DOD had a noticeable impact on the degradation process of all three technologies, especially at 0°C and 25°C cycling aging. It could also be observed that (i) NMC/LTO technology is a superior choice for applications that require a standard C-rate, stable performance, longer battery life, and higher delivered capacity; (ii) For high C-rate cycling, the choice of technology and ambient temperature play a critical role in determining the long-term performance and stability of the battery. NMC/LTO was found to be the better choice for stability at elevated temperatures, while NMC/Gr performed better at intermediate temperatures of 25°C; (iii) Calendar aging and cycling at elevated temperature (50°C) resulted in faster degradation for LFP/Gr compared to NMC cells, making NMC a better choice for high temperature applications.

Subproject T3.7 was complementary to T3.6. While batteries are good at storing large amounts of energy but can be easily stressed by high charging or discharging power, supercapacitors can deliver at least an order of magnitude higher power at exceptional life of close to 100,000 cycles – but they are generally limited in their energy storage capacity. State-of-the-art commercial supercapacitors rely on the physical mechanism of charge storage, called electrical double layer capacitance (EDLCs). Owing

to these characteristics, the use of supercapacitors can be beneficial to help mitigate limitations of the current battery technologies related to the slow charging and limited cycle life through tandem integration. To realize this benefit, the characteristics of commercial supercapacitors must be properly understood. Therefore, the project aimed at: (i) conducting a comprehensive electrochemical behavior assessment of commercial supercapacitors, (ii) identifying critical parameters for maintaining stable cycling performance, (iii) and evaluating the potential integration of supercapacitors into battery-based grid storage systems [27]. The results showed that even with similar specifications, supercapacitor cells from different manufacturers can degrade at markedly different rates, stressing the need for independent testing prior to their use in grid storage. A notable finding is the approximately 5% cell-to-cell capacity variation, which can cause uneven current and voltage distribution when cells are assembled into modules, potentially leading to operation beyond manufacturer-recommended conditions. Furthermore, cells subjected to just slightly above recommended voltage degrade at a significantly faster rate. The study also highlighted the detrimental effect of high currents on capacity retention, a situation that worsens with increased temperature. Thermal imaging has shown that high current operation leads to substantial rises in cell temperature, pointing to the necessity of efficient thermal management in module design. Lastly, the integration of battery-supercapacitor modules has demonstrated potential in enhancing battery life by minimizing stress through narrower state-of-charge ranges, offering a promising direction for extending the longevity of energy storage systems.

The full functionality of the ReMaP platform was used for a total of four HIL experiments in three Subprojects. Even more projects used ReSIM, the simulation framework. Both the hardware components and improved control capabilities that were built up at the two experimental platforms ESI and ehub live on after closure of ReMaP. In addition, ReSIM forms the basis for the analysis of the flexibility of multi-energy systems in the SFOE-sponsored SWEET project PATHFNDR.

Zusammenfassung

Die Schweiz hat sich zum Ziel gesetzt, bis 2050 netto null Treibhausgasemissionen zu erreichen. Verschiedene Teams haben mithilfe von Energiesystemmodellen machbare Pfade beschrieben, was zu einem Konsens über viele entscheidende Aspekte führte. Die wichtigsten sind die Elektrifizierung des Mobilitäts- und des Wärmesektors durch batteriebetriebene Elektrofahrzeuge bzw. Wärmepumpen und die dominante Rolle der Photovoltaik im zukünftigen Energiemix. Die Herausforderungen, die mit dieser Umstellung von einem System mit fossilen Brennstoffen auf ein elektrisches System mit volatiler Erzeugung einhergehen, liegen auf der Hand. Es ist daher von entscheidender Bedeutung, sich mit diesen Herausforderungen auseinanderzusetzen und zu verstehen, welche Auswirkungen die breite Einführung fluktuierender erneuerbarer Energiequellen auf das Energiesystem insgesamt und insbesondere auf das Verteilungsnetz hat.

Das Projekt "Renewable Management and Real-Time Control Platform" (ReMaP) trug zu diesem Verständnis bei, indem es eine flexible, hardware- und softwarebasierte Forschungsplattform zur Bewertung potenzieller Energiesystemlösungen für zukünftige Quartiere entwickelte. Diese Plattform ermöglicht es, Multi-Energie-Systeme auf der Verteilnetzebene zu testen, zu analysieren und zu optimieren und damit die Zusammenarbeit zwischen Wissenschaft und Industrie zu fördern. Die Plattform zeichnet sich dadurch aus, dass sie die bestehenden Plattformen ESI (Energy System Integration) des Paul Scherrer Instituts sowie die Plattformen NEST, move und ehub der Empa integriert.

Das ReMaP-Projekt basiert auf zwei Säulen: Die erste ist die ReMaP-Plattform selbst, die das erwähnte Testen zukünftiger Multi-Energie-Systeme ermöglicht - von einer reinen Computersimulation, d.h. der ReSIM-Software, bis hin zu einem Hardware-in-the-Loop (HIL)-Ansatz, bei dem einige oder alle Elemente durch reale Hardware dargestellt werden. Zum anderen gibt es insgesamt neun Unterprojekte, die sich auf spezifische Aspekte der Herausforderungen beim Aufbau und der Steuerung eines zukünftigen Energiesystems konzentrieren. Die meisten dieser Unterprojekte nutzten Elemente der ReMaP-Plattform, viele bereicherten die Plattform, z.B. durch die Bereitstellung neuartiger Steuerungsansätze in Form von Softwaremodulen, einige führten vollständige HIL-Experimente durch, die ESI, Nest und sogar zusätzliche Elemente einschlossen, wie z.B. ein Blockheizkraftwerk an der ETH oder eine Biogasanlage in Inwil. Dieser Bericht dient als Leitfaden zu den Ergebnissen, die während des Projekts erzielt wurden.

Die Teilprojekte T3.1-10 deckten eine Reihe sich überschneidender Themen ab, die sich wie folgt gruppieren lassen: (i) innovative Steuerungs- und Designansätze (T3.1, T3.3), (ii) Umgang mit der Unsicherheit von Lasten (T3.2, T3.4), (iii) Batterie- und Brennstoffzellen-Elektrofahrzeuge (T3.9, T3.10) und (iv) verbesserte Komponenten (T3.5, T3.6, T3.7).

Innovative Steuerungs- und Designansätze: Die Einführung erneuerbarer Energiequellen und flexibler Nachfrage, insbesondere in Verteilnetzen, eröffnet neue Wege zu nachhaltigeren Energiesystemen. Die Volatilität und Unvorhersehbarkeit dieser Elemente, z. B. der Nachfrage der Haushalte oder der verfügbaren erneuerbaren Energie, stellen jedoch eine große Herausforderung für die Zuverlässigkeit des Netzes dar. Daher ist ein schnellerer und effizienterer Netzbetrieb für die Schaffung eines nachhaltigeren Stromsystems unerlässlich. Das **Teilprojekt T3.1** untersuchte verschiedene neue Regelungsansätze, die auf Online-Feedback-basierter Optimierung (OFO) basieren. Simulationen mit Standardtestfällen bewiesen ihre Effektivität in Fällen mit verrauschten und unvollständigen Messungen, insbesondere durch die Kombination von OFO mit einer dynamischen Schätzung, die auf linearen und nichtlinearen Leistungsflussmodellen basiert. Es wurde auch ein modellfreies OFO untersucht, das die Input-Output-Empfindlichkeit lernt. Dies wurde durch die Kombination eines OFO-Standardreglers mit einer rekursiven Least-Squares-Schätzung erreicht, um die Empfindlichkeit aus Messungen während des Echtzeitbetriebs zu lernen. Auch hier wurde die Leistung des Ansatzes anhand einer numerischen Simulation mit dem IEEE-123-Bus-Test-Feeder validiert und zeigte eine überlegene Leistung im Vergleich zu einem OFO nach dem Stand der Technik mit einer konstanten Empfindlichkeitsnäherung [1].

Während sich die zuvor beschriebenen Ergebnisse hauptsächlich auf die Steuerung der Verteilnetzebene bezogen, ist die Interaktion dieser Niederspannungsebenen mit den höheren

Übertragungsnetzebenen von größter Bedeutung. Das **Teilprojekt T3.3** war dieser Verbindung gewidmet und konzentrierte sich auf aktive elektrische Verteilungsnetze (ADN) und die Interaktion mit ihrem Hauptnetz. Dies wurde unter Verwendung der ReSIM-Simulationsumgebung, einschließlich der Adaptricity-Netzsimulationssoftware, untersucht. Der Schwerpunkt lag dabei auf den Auswirkungen von ADNs und Elektrifizierung auf Spannungsabweichungen, Leitungs- und Transformatorbelastung in einem ländlichen, einem halbstädtischen und einem städtischen Stromnetz. Während die zusätzliche Last aufgrund der Elektrifizierung von Heizung und Mobilität in ländlichen Netzen zu Spannungseinschränkungen und in halbstädtischen und städtischen Netzen zu kurzfristigen Überlastungen von Leitungen führen kann, kann die Umstellung von 50 % der Lastbusse auf optimal ausgelegte ADNs die Spannungs- und Überlastprobleme in allen untersuchten Netzen mit zwei sich ergänzenden Mechanismen lösen: (i) Die verteilten Energieressourcen (DER) innerhalb der ADNs ermöglichen einen höheren Eigenverbrauch, wodurch die Gesamtlast im Netz reduziert wird. (ii) In ReSIM wurde ein Energiemanagementsystem entwickelt und getestet, das die verfügbare DER-Kapazität nutzt, um das durch Day-Ahead-Prognosefehler verursachte Ungleichgewicht zwischen Last und Erzeugung auszugleichen und Spannungs- und Überlastungsprobleme in den übergeordneten Netzen zu lindern [14].

Umgang mit der Unsicherheit von Lasten: Für einen besseren Netzbetrieb ist die Nutzung kurzfristiger Vorhersagen, sei es Minuten oder Tage im Voraus, auf Kundenebene von großem Nutzen. Die über intelligente Zähler und Gerätesensoren gesammelten granularen Daten können als Input für maschinelle Lernverfahren verwendet werden, die dann Vorhersagen für den gewünschten Zeitraum liefern. Auf der Übertragungsebene können deterministische Prognosealgorithmen die Last, die relativ gleichmäßig und periodisch ist, richtig vorhersagen. Die Last in Verteilnetzen, insbesondere auf der Kundenebene, ist jedoch sehr volatil, so dass deterministische Ansätze nicht zufriedenstellend sind. Aufgrund der inhärenten Unsicherheiten und Ungenauigkeiten von Vorhersagen sollte der Ansatz daher probabilistische Vorhersagen liefern, d. h. Verteilungen und Wahrscheinlichkeitsquantile anstelle von deterministischen Werten. Das **Teilprojekt T3.2** nahm sich dieses Problems an, indem es probabilistische Vorhersagealgorithmen entwickelte, die tatsächlich den gesamten Unsicherheitsbereich abdecken können, ohne von bestimmten Fehlerverteilungen auszugehen, und die ihre Vorhersage in Echtzeit aktualisieren können. Um dies zu erreichen, wurden historische Daten verwendet, um mehrere Regressionsmodelle zu trainieren, die unabhängige Vorhersagen generieren. Diese Modelle wurden dann zu einem Ensemble-Modell kombiniert, um die Genauigkeit zu erhöhen. Sowohl deterministische als auch probabilistische Modelle wurden sowohl auf Gebäude- als auch auf Einzelgeräteebene angewendet. Die Ergebnisse zeigten, dass deterministische Modelle aufgrund der Volatilität der Last weniger effektiv sind, während probabilistische Modelle bessere Unsicherheitsintervalle liefern. Dieser Ansatz wurde als Softwarepaket in ReSIM integriert, um Stromflüsse auf Gebäudeebene vorherzusagen [6].

Die Kopplung des Wärme- und Stromsektors ist eine Voraussetzung, um die Schweizer Klimaziele zu erreichen. Während Wärmepumpen offensichtlich die Technologie der Zukunft sind, erhöht ihr weit verbreiteter Einsatz die Unsicherheit des zukünftigen Strombedarfs, sowohl auf jährlicher Ebene als auch auf kürzeren Zeitskalen. Das **Teilprojekt T3.4** befasste sich mit dem Aspekt der Gebäude und ihrer Interaktion mit zukünftigen Energiesystemen [21]. Drei Stadtteil-Archetypen (städtisch, vorstädtisch und ländlich) wurden definiert und mit dem City Energy Analyst (CEA) simuliert, einem Open-Source-Rechenprogramm für die Analyse städtischer Energiesysteme. CEA verwendet dynamische und multiphysikalische Modelle zur Vorhersage des Lastprofils von Heizung, Kühlung und Strom für einzelne und mehrere Gebäude bis hin zu ganzen Stadtteilen in stündlichen Zeitschritten. Darüber hinaus wurde ein Plug-in für die Transformation des Gebäudebestands entwickelt, um Eingangsszenarien für die Energiebedarfssimulation zu erzeugen. Die Ergebnisse zeigen, dass im Jahr 2060 im Archetyp eines Stadtviertels der Bedarf an Raumkühlung die gleiche Höhe wie der Bedarf an Raumwärme erreichen kann, was zu wichtigen Überlegungen hinsichtlich der Systemauslegung und -optimierung in diesen Gebieten führt - insbesondere in Verbindung mit dem Stromsystem. Im Archetyp des suburbanen Stadtteils haben die Ergebnisse gezeigt, dass sich die Energieplanung als sehr herausfordernd erweisen kann, abhängig von den städtebaulichen Entwicklungspfaden und den Energieeffizienzstrategien. Im Gegensatz dazu erwiesen sich diese Schwierigkeiten im Archetyp des ländlichen Bezirks als weniger ausgeprägt, wobei der Raumwärmebedarf den Entscheidungsprozess möglicherweise dominiert, jedoch stark vom lokalen regionalen Klima abhängt.

Batterie- und Brennstoffzellen-Elektrofahrzeuge: Das **Teilprojekt T3.9** zielte darauf ab, fortschrittliche Steuerungsalgorithmen für den optimalen Betrieb zukünftiger Energiesysteme zu entwickeln und zu analysieren, wobei ein besonderer Schwerpunkt auf der Integration zwischen dem Energie- und dem Mobilitätssektor lag. Es wurde eine Modellierungsbibliothek mit detaillierten Modellen der Geräte auf der ESI-Plattform und auf der move-Plattform entwickelt, die die Nichtlinearitäten realer Hardwarekomponenten erfassen und letztlich die Entwicklung genauer modellbasierter Simulationen und Steuerungsschemata unterstützen kann. Zu diesen Geräten gehören Elektrolyseure, Brennstoffzellen, Gasreiniger, Kompressoren und Lagertanks. Im Rahmen des Projekts wurden auch fortschrittliche Steuerungsalgorithmen für die optimale Steuerung künftiger Energieknotenpunkte entwickelt, die mit der inhärenten Stochastik solcher Systeme umgehen können und über sehr gute Eigenschaften in Bezug auf Robustheit, Rechenaufwand und Skalierbarkeit verfügen. Es wurde ein Online-Optimierungsalgorithmus entwickelt, der ein Optimierungsproblem für die Echtzeitsteuerung komplexer Systeme löst. Das Hauptaugenmerk lag dabei auf dem rechnerischen Aspekt: Es wurde ein duales dynamisches Programmierungsverfahren für die Steuerung nichtlinearer Probleme über einen langen Zeitraum (z. B. für die saisonale Energiespeicherung) mit einem optimalen Kompromiss zwischen Genauigkeit und Rechenaufwand entwickelt. Ein weiteres Beispiel war ein Online-Peak-Shaving-Optimierungsregler, der es ermöglichte, die Spitzenlast einer mit einem Wasserstoff-Energiespeichersystem ausgestatteten EV-Ladestation zu reduzieren. Diese Steuerung wurde mit der ReMaP-Plattform in einem HIL-Experiment auf der ESI-Plattform des PSI getestet. Diese Tests bestätigten, dass die Leistung bei der Begrenzung der von der Ladestation gemessenen Spitzenleistung besser ist als bei bestehenden Regelsystemen [30].

Ein sehr ähnliches Problem einer Schnellladestation an einer Autobahn wurde mit HIL-Experimenten im **Teilprojekt T3.10** untersucht. Auch in diesem Fall konnte gezeigt werden, wie die Kombination mit einem Wasserstoffspeicher zur Minimierung der Spitzenleistung beitragen kann. In einem zweiten Fall wurde die netzneutrale und regenerative Wasserstofferzeugung für eine Brennstoffzellenfahrzeugflotte betrachtet. Hier wurde ein Elektrolyseur zusammen mit einem saisonalen Power-to-Gas-Speichersystem betrachtet, das aus einer Methanisierung, dem Gasnetz als Zwischenspeicher und einem Dampfreformer besteht. Es wurde der dynamische Betrieb der Anlage über das Jahr simuliert. Die Simulationsstudie zeigte, dass in den nächsten Jahrzehnten weiter sinkende Kosten für die Elektrolyse und die PV-Anlage sowie steigende Wasserstoffpreise die netzneutrale regenerative Wasserstofferzeugung wirtschaftlich machen können. Für die experimentelle Kampagne wurden die Elektrolyse, die Gasreinigung und der H2-Tank am PSI, die containerbasierte Methanisierungseinheit, die an eine reale Biogasanlage in Inwil angeschlossen ist, sowie die Batterie an der Empa, die alle über das ReMaP-Framework verbunden sind, für ein HIL-Experiment verwendet. Es konnte gezeigt werden, dass die Methanisierungsanlage in der Lage ist, Teillastwechsel zu bewältigen, ohne die Gasqualität zu beeinträchtigen [39].

Verbesserte Komponenten: Das ReMaP-Projekt betonte den Systemaspekt zukünftiger Energiesysteme, dennoch besteht ein System aus Komponenten, von denen drei in den **Teilprojekten T3.5** (Kraft-Wärme-Kopplungsanlagen, KWK), **T3.6** (Batterien) und **T3.7** (Superkondensatoren) detailliert untersucht wurden. KWK-Anlagen wandeln chemische Energie in thermische und elektrische Energie um. Solche Anlagen können schnell in Betrieb genommen werden und haben ein großes Potenzial, zu einem stabilen zukünftigen Energiesystem beizutragen, das weitgehend auf schwankenden erneuerbaren Energiequellen basiert. Daher bestand das übergeordnete Ziel des **Teilprojekts T3.5** darin, einen Weg zu finden, sowohl die Exergieeffizienz als auch die Flexibilität solcher Anlagen zu erhöhen [25]. Zusätzlich wurde die Fähigkeit zur saisonalen Energiespeicherung untersucht. Der Schwerpunkt lag auf Mikro-KWK-Anlagen mit einer elektrischen Leistung von unter 10 kW, aber die entwickelten Konzepte sollten bis zu einem gewissen Grad auch für größere Anlagen geeignet sein. Zwei Optionen zur Verbesserung der Flexibilität und der Exergieeffizienz eines konventionellen KWK-Systems wurden mit Hilfe von Simulationen untersucht. Die vielversprechendere Option wurde mit Hardware-in-the-Loop (HIL)-Experimenten auf der ReMaP-Plattform weiter getestet (siehe auch Abschnitt 3.1). Es zeigte sich, dass ein Dampf-Methan-Reformer (SMR) KWK-Konzept gegenüber dem Konzept der Kombination der KWK mit einem Erdwärmespeicher und einer Wärmepumpe zu bevorzugen ist. Ein solches System verspricht, flexibler (Erhöhung des Autarkiegrades um 14,0 Prozentpunkte) und exergetisch effizienter (Erhöhung um 2,9 Prozentpunkte) zu sein als ein konventionelles System.

Batterien werden eine Schlüsselkomponente künftiger Energiesysteme sein - in batteriebetriebenen Elektrofahrzeugen und als stationäre Stromspeicher. Das Ziel des **Teilprojekts T3.6** war es, Erkenntnisse über die Faktoren zu gewinnen, die zur Degradation von Batterien beitragen, und die Langlebigkeit von Batterien für stationäre Anwendungen zu verbessern [26]. Zu diesem Zweck wurde eine zyklische Alterungskampagne durchgeführt, um reale Bedingungen zu simulieren, einschließlich Standard- und "Missbrauchsszenarien". Getestet wurden vier verschiedene Temperaturen, zwei verschiedene Lade-/Entladeraten und drei verschiedene Entladetiefen (Depth of Discharge, DOD). Während des Alterungsprozesses wurden verschiedene Parameter überwacht, wobei die wichtigsten Parameter der Effizienzverlust pro Zyklus und der Verlust der Entladekapazität waren. Beide sind nützlich, um die Lebensdauer der Batteriezelle vorherzusagen. Die wichtigsten Ergebnisse waren, dass die Temperatur einen starken Einfluss auf die Leistung aller drei untersuchten Zelltechnologien (NMC/LTO, NMC/Gr und LFP/Gr) hatte. DOD hatte einen spürbaren Einfluss auf den Degradationsprozess aller drei Technologien, insbesondere bei einer Alterung bei 0°C und 25°C. Es konnte auch festgestellt werden, dass (i) die NMC/LTO-Technologie die bessere Wahl für Anwendungen ist, die eine Standard-C-Rate, eine stabile Leistung, eine längere Batterielebensdauer und eine höhere gelieferte Kapazität erfordern; (ii) bei Zyklen mit hoher C-Rate spielen die Wahl der Technologie und die Umgebungstemperatur eine entscheidende Rolle bei der Bestimmung der langfristigen Leistung und Stabilität der Batterie. NMC/LTO erwies sich als die bessere Wahl für die Stabilität bei höheren Temperaturen, während NMC/Gr bei Zwischentemperaturen von 25°C besser abschnitten; (iii) Kalenderalterung und Zyklen bei höheren Temperaturen (50°C) führten zu einer schnelleren Degradation von LFP/Gr-Zellen im Vergleich zu NMC-Zellen, was NMC zu einer besseren Wahl für Hochtemperaturanwendungen macht.

Das **Teilprojekt T3.7** war eine Ergänzung zu T3.6. Während Batterien gut geeignet sind, große Energiemengen zu speichern, aber durch hohe Lade- oder Entladeleistungen leicht belastet werden können, können Superkondensatoren mindestens eine Größenordnung mehr Leistung bei einer außergewöhnlichen Lebensdauer von fast 100.000 Zyklen liefern - aber sie sind im Allgemeinen in ihrer Energiespeicherkapazität begrenzt. Der Stand der Technik bei kommerziellen Superkondensatoren beruht auf dem physikalischen Mechanismus der Ladungsspeicherung, der so genannten elektrischen Doppelschichtkapazität (EDLCs). Aufgrund dieser Eigenschaften kann der Einsatz von Superkondensatoren dazu beitragen, die Einschränkungen der derzeitigen Batterietechnologien, die mit der langsamen Aufladung und der begrenzten Zyklusdauer zusammenhängen, durch Tandemintegration zu mildern. Um diese Vorteile zu nutzen, müssen die Eigenschaften kommerzieller Superkondensatoren richtig verstanden werden. Daher zielte das Projekt auf Folgendes ab: (i) eine umfassende Bewertung des elektrochemischen Verhaltens handelsüblicher Superkondensatoren, (ii) die Ermittlung kritischer Parameter für die Aufrechterhaltung einer stabilen Zyklusleistung, (iii) und die Bewertung der potenziellen Integration von Superkondensatoren in batteriegestützte Netzspeichersysteme [27]. Die Ergebnisse zeigten, dass sich Superkondensatorzellen verschiedener Hersteller trotz ähnlicher Spezifikationen sehr unterschiedlich schnell abbauen können, was die Notwendigkeit unabhängiger Tests vor ihrem Einsatz in Netzspeichern unterstreicht. Ein bemerkenswertes Ergebnis ist die Kapazitätsabweichung von etwa 5 % von Zelle zu Zelle, die zu einer ungleichmäßigen Strom- und Spannungsverteilung führen kann, wenn die Zellen zu Modulen zusammengebaut werden, was möglicherweise zu einem Betrieb außerhalb der vom Hersteller empfohlenen Bedingungen führt. Darüber hinaus verschlechtern sich Zellen, die nur geringfügig über der empfohlenen Spannung liegen, deutlich schneller. In der Studie wurde auch die nachteilige Wirkung hoher Ströme auf die Kapazitätserhaltung hervorgehoben, eine Situation, die sich mit steigender Temperatur noch verschlimmert. Wärmebildaufnahmen haben gezeigt, dass der Betrieb mit hohen Strömen zu einem erheblichen Anstieg der Zelltemperatur führt, was auf die Notwendigkeit eines effizienten Wärmemanagements bei der Modulkonstruktion hinweist. Schließlich hat sich gezeigt, dass die Integration von Batterie-Superkondensator-Modulen das Potenzial hat, die Lebensdauer von Batterien zu verlängern, indem die Belastung durch engere Ladezustandsbereiche minimiert wird, was einen vielversprechenden Weg zur Verlängerung der Lebensdauer von Energiespeichersystemen darstellt.

Die volle Funktionalität der ReMaP-Plattform wurde für insgesamt vier HIL-Experimente in drei Teilprojekten genutzt. In noch mehr Projekten wurde das Simulationsframework ReSIM eingesetzt. Sowohl die Hardwarekomponenten als auch die verbesserten Steuerungsmöglichkeiten, die an den

beiden Experimentierplattformen ESI und ehub aufgebaut wurden, leben nach Abschluss von ReMaP weiter. Zudem bildet ReSIM die Grundlage für die Analyse der Flexibilität von Multi-Energie-Systemen im BFE-geförderten SWEET-Projekt PATHFNDR.

Resumé

La Suisse s'est fixé pour objectif d'atteindre des émissions nettes de gaz à effet de serre de zéro d'ici 2050. De nombreuses équipes ont utilisé des modèles de systèmes énergétiques pour décrire les voies possibles, ce qui a permis de dégager un consensus sur de nombreux aspects cruciaux. Les plus importants sont l'électrification des secteurs de la mobilité et du chauffage par le biais de véhicules électriques à batterie et de pompes à chaleur, respectivement, et le rôle dominant de l'énergie photovoltaïque dans le futur bouquet énergétique. Les défis qui accompagnent cette transformation d'un système à base de combustibles fossiles en un système électrique reposant sur une production volatile sont évidents. Il est donc essentiel de relever ces défis et de comprendre les implications de l'adoption généralisée de sources d'énergie renouvelables fluctuantes sur le système énergétique dans son ensemble et en particulier sur le réseau de distribution.

Le projet "Renewable Management and Real-Time Control Platform" (ReMaP) a contribué à cette compréhension en développant une plateforme de recherche flexible, basée sur du matériel et des logiciels, pour évaluer les solutions potentielles de systèmes énergétiques pour le quartier du futur. Cette plateforme permet de tester, d'analyser et d'optimiser des systèmes multi-énergies au niveau de la distribution et d'encourager ainsi la collaboration entre le monde universitaire et l'industrie. Elle se distingue par le fait qu'elle intègre la plate-forme d'intégration des systèmes énergétiques de l'Institut Paul Scherrer et les plates-formes NEST, move et ehub de l'Empa.

Le projet ReMaP repose sur deux piliers : le premier est la plate-forme ReMaP elle-même, qui permet de tester les futurs systèmes multi-énergies - d'une simulation informatique pure, c'est-à-dire le logiciel ReSIM, à une approche hardware-in-the-loop (HIL) où tout ou partie des éléments sont représentés par du matériel réel. Le second est un total de neuf sous-projets qui se concentrent sur des aspects spécifiques des défis à relever pour construire et contrôler un futur système énergétique. La plupart de ces sous-projets ont utilisé des éléments de la plateforme ReMaP, beaucoup ont enrichi la plateforme, par exemple en fournissant de nouvelles approches de contrôle sous la forme de modules logiciels, certains ont réalisé des expériences HIL complètes impliquant ESI, Nest et même des éléments supplémentaires, tels qu'une centrale de production combinée de chaleur et d'électricité à l'ETH ou une centrale de biogaz à Inwil. Ce rapport sert de guide pour les résultats obtenus au cours du projet.

Les sous-projets T3.1-10 couvrent un certain nombre de thèmes qui se recoupent et qui peuvent être regroupés comme suit : (i) approches innovantes en matière de contrôle et de conception (T3.1, T3.3), (ii) gestion de l'incertitude des charges (T3.2, T3.4), (iii) véhicules électriques à batterie et à pile à combustible (T3.9, T3.10), et (iv) composants améliorés (T3.5, T3.6, T3.7).

Approches innovantes en matière de contrôle et de conception : L'introduction de sources d'énergie renouvelables et d'une demande flexible, en particulier dans les réseaux de distribution, ouvre de nouvelles voies vers des systèmes électriques plus durables. Cependant, la volatilité et l'imprévisibilité de ces éléments, par exemple la demande des ménages ou l'énergie renouvelable disponible, posent de graves problèmes de fiabilité du réseau. Par conséquent, un fonctionnement plus rapide et plus efficace du réseau est essentiel pour créer un système électrique plus durable. Le **sous-projet T3.1** a exploré différentes nouvelles approches de contrôle basées sur l'optimisation en ligne basée sur le retour d'information (OFO). Des simulations avec des cas de test standard ont prouvé leur efficacité dans les cas de mesures bruyantes et incomplètes, en particulier en combinant l'OFO avec une estimation dynamique basée sur des modèles de flux de puissance linéaires et non linéaires. Une OFO sans modèle a également été étudiée, qui apprend la sensibilité entrée-sortie. Ceci a été réalisé en combinant un contrôleur OFO standard avec une estimation récursive des moindres carrés pour apprendre la sensibilité à partir de mesures pendant le fonctionnement en temps réel. Une fois encore, les performances de l'approche ont été validées par une simulation numérique utilisant la ligne d'alimentation IEEE 123-bus, montrant des performances supérieures à celles d'un OFO de pointe avec une approximation de sensibilité constante [1].

Bien que les résultats décrits précédemment concernent principalement le contrôle du niveau du réseau de distribution, l'interaction de ces niveaux de basse tension avec les niveaux supérieurs du réseau de transmission est d'une importance capitale. Le **sous-projet T3.3** a été consacré à ce lien en se concentrant sur les réseaux de distribution électrique actifs (ADN) et l'interaction avec leur réseau

d'accueil. Cette étude a été réalisée à l'aide du cadre de simulation ReSIM, y compris le logiciel de simulation de réseau Adaptricity, en se concentrant sur l'impact des ADN et de l'électrification sur les déviations de tension, la charge des branches et des transformateurs dans un réseau d'accueil rural, semi-urbain et urbain. Alors que la charge supplémentaire due à l'électrification du chauffage et de la mobilité peut conduire à des violations des contraintes de tension dans les réseaux ruraux et à des surcharges de branche à court terme dans les réseaux semi-urbains et urbains, la transition de 50 % des bus de charge vers des ADN conçus de manière optimale peut résoudre les problèmes de tension et de surcharge dans tous les réseaux étudiés grâce à deux mécanismes complémentaires : (i) les ressources énergétiques distribuées (DER) au sein des ADN permettent une autoconsommation plus élevée, ce qui réduit la charge totale dans le réseau. (ii) Un système de gestion de l'énergie a été développé et testé dans ReSIM, qui utilise la capacité des DER disponibles pour atténuer le déséquilibre entre la charge et la production causé par les erreurs de prévision du jour précédent et pour atténuer les problèmes de tension et de surcharge dans les réseaux d'hébergement [14].

Gestion de l'incertitude des charges : Pour un meilleur fonctionnement du réseau, l'utilisation de prévisions à court terme, que ce soit quelques minutes ou quelques jours à l'avance, au niveau du client est d'une grande utilité. Les données granulaires collectées par les compteurs intelligents et les capteurs d'appareils peuvent être utilisées comme données d'entrée pour les approches d'apprentissage automatique qui fournissent alors des prévisions pour la plage de temps souhaitée. Au niveau de la transmission ou à un niveau très agrégé, les algorithmes de prévision déterministes peuvent prédire correctement la charge qui est relativement régulière et périodique. Cependant, la charge dans les réseaux de distribution, et en particulier au niveau du client, est très volatile, de sorte que les approches déterministes ne sont pas satisfaisantes. Par conséquent, en raison des incertitudes et des imprécisions inhérentes aux prédictions, l'approche devrait fournir des prédictions probabilistes, c'est-à-dire des distributions et des quantiles de probabilité au lieu de valeurs déterministes. Le **sous-projet T3.2** s'est attaqué à ce problème en développant des algorithmes de prévision probabiliste qui peuvent en effet couvrir l'ensemble de la plage d'incertitude sans supposer de distributions d'erreur spécifiques et qui peuvent mettre à jour leurs prévisions en temps réel. Pour ce faire, des données historiques ont été utilisées pour former des modèles de régression multiples, générant des prévisions indépendantes. Ces modèles ont ensuite été combinés en un modèle d'ensemble pour une meilleure précision. Des modèles déterministes et probabilistes ont été appliqués au niveau des bâtiments et des appareils individuels. Les résultats ont montré que les modèles déterministes sont moins efficaces en raison de la volatilité de la charge, tandis que les modèles probabilistes fournissent de meilleurs intervalles d'incertitude. Cette approche a été intégrée en tant que logiciel dans ReSIM pour prédire les flux d'électricité au niveau des bâtiments [6].

Le couplage des secteurs du chauffage et de l'électricité est une condition préalable pour atteindre les objectifs climatiques de la Suisse. Si les pompes à chaleur sont manifestement la technologie de l'avenir, leur utilisation généralisée accroît l'incertitude quant à la demande future d'électricité, au niveau annuel mais aussi sur des échelles de temps plus courtes. Le **sous-projet T3.4** a traité de l'aspect des bâtiments et de leur interaction avec les futurs systèmes énergétiques [21]. Trois archétypes de quartiers (urbain, suburbain et rural) ont été définis et simulés à l'aide du City Energy Analyst (CEA), un cadre informatique libre pour l'analyse des systèmes énergétiques urbains. Le CEA utilise des modèles dynamiques et multi-physiques pour prévoir le profil de charge du chauffage, de la climatisation et de l'électricité d'un ou de plusieurs bâtiments jusqu'à des quartiers entiers, au pas de temps horaire. En outre, un module d'extension pour la transformation du parc immobilier a été développé pour générer des scénarios d'entrée pour la simulation de la demande d'énergie. Les résultats ont montré que d'ici 2060, dans l'archétype du quartier urbain, la demande de refroidissement des locaux peut atteindre des quantités équivalentes à la demande de chauffage des locaux, ce qui conduit à des considérations importantes en ce qui concerne la conception et l'optimisation des systèmes dans ces zones - en particulier en ce qui concerne le système électrique. Dans l'archétype du quartier suburbain, les résultats ont révélé que la planification énergétique peut s'avérer très difficile, en fonction des trajectoires de développement urbain et des stratégies d'efficacité énergétique. À l'inverse, dans l'archétype du district rural, ces difficultés se sont révélées moins prononcées, la demande de chauffage des locaux pouvant dominer le processus de prise de décision, tout en dépendant fortement du climat régional local.

Véhicules électriques à batterie et à pile à combustible : Le **sous-projet T3.9** visait à développer et à analyser des algorithmes de contrôle avancés pour l'exploitation optimale des futurs systèmes énergétiques, en mettant l'accent sur l'intégration entre le secteur de l'énergie et celui de la mobilité. Une bibliothèque de modélisation avec des modèles détaillés des dispositifs de la plateforme ESI et de la plateforme move a été développée pour capturer les non-linéarités des composants matériels du monde réel et soutenir en fin de compte le développement de simulations et de schémas de contrôle précis basés sur des modèles. Ces dispositifs comprenaient des électrolyseurs, des piles à combustible, des épurateurs de gaz, des compresseurs et des réservoirs de stockage. Le projet a également fourni des algorithmes de contrôle avancés pour le contrôle optimal des futurs centres énergétiques, capables de gérer la stochasticité inhérente à ces systèmes et accompagnés de certificats en termes de robustesse, d'empreinte de calcul et d'évolutivité. Un algorithme d'optimisation en ligne a été développé pour résoudre un problème d'optimisation pour le contrôle en temps réel de systèmes complexes. L'accent a été mis ici sur l'aspect informatique : un cadre de programmation dynamique duale a été fourni pour le contrôle de problèmes non linéaires à long terme (par exemple, pour le stockage saisonnier de l'énergie) avec un compromis optimal entre la précision et le calcul. Un autre exemple est celui d'un contrôleur d'optimisation de l'écrêtement des pointes en ligne qui permet d'écrêter les pointes d'une station de recharge de véhicules électriques équipée d'un système de stockage d'énergie à l'hydrogène. Ce contrôleur a été testé à l'aide de la plateforme ReMaP dans le cadre d'une expérience HIL sur la plateforme ESI du PSI. Ces tests ont confirmé que les performances étaient supérieures aux schémas de contrôle existants pour limiter la puissance de pointe de la station de charge [30].

Un problème très similaire d'une station de recharge rapide sur une autoroute a été étudié avec des expériences HIL dans le cadre du **sous-projet T3.10**. Dans ce cas également, il a été possible de montrer comment la combinaison avec un stockage d'hydrogène peut contribuer à minimiser la puissance de pointe de la charge. Un deuxième cas concernait la production d'hydrogène renouvelable et neutre par rapport au réseau pour un parc de véhicules à pile à combustible. Dans ce cas, un électrolyseur a été envisagé avec un système de stockage saisonnier Power-to-Gas, qui consiste en une méthanisation, le réseau de gaz comme stockage intermédiaire et un reformeur à vapeur. Le fonctionnement dynamique de l'installation sur l'année a été simulé. L'étude de simulation a montré qu'au cours des prochaines décennies, la baisse des coûts de l'électrolyse et du système photovoltaïque ainsi que l'augmentation du prix de l'hydrogène rendront la production d'hydrogène renouvelable neutre pour le réseau économiquement réalisable. Pour la campagne expérimentale, l'électrolyse, l'épuration des gaz et le réservoir d'hydrogène du PSI, l'unité de méthanisation en conteneur connectée à une véritable usine de biogaz à Inwil ainsi que la batterie de l'Empa, tous connectés via le cadre ReMaP, ont été utilisés pour une expérience HIL. Il a été démontré que l'unité de méthanisation est capable de gérer des changements de charge partiels sans compromettre la qualité du gaz [39].

Composants améliorés : Le projet ReMaP a mis l'accent sur l'aspect systémique des futurs systèmes énergétiques, mais un système est constitué de composants, dont trois ont été étudiés en détail dans les **sous-projets T3.5** (centrales de production combinée de chaleur et d'électricité, PCCE), **T3.6** (batteries) et **T3.7** (supercondensateurs). Les centrales de cogénération convertissent l'énergie chimique en énergie thermique et électrique. Ces installations peuvent être mises en service rapidement et ont un grand potentiel pour contribuer à un futur système énergétique stable basé en grande partie sur des sources d'énergie renouvelables fluctuantes. Par conséquent, l'objectif global du **sous-projet T3.5** était de trouver un moyen d'accroître à la fois l'efficacité énergétique et la flexibilité de ces centrales [25]. En outre, la capacité de stocker l'énergie de manière saisonnière a été étudiée. L'accent a été mis sur les centrales de micro-cogénération d'une puissance de sortie électrique inférieure à 10 kW, mais les concepts développés devraient également pouvoir convenir à des centrales plus importantes dans une certaine mesure. Deux options pour améliorer la flexibilité et l'efficacité énergétique d'un système de cogénération conventionnel ont été étudiées à l'aide de simulations. L'option la plus prometteuse a été testée plus avant avec des expériences Hardware-in-the-Loop (HIL) sur la plateforme ReMaP (voir également la section 3.1). Il a été constaté qu'un concept de cogénération par reformage du méthane à la vapeur (SMR) est à privilégier par rapport au concept consistant à combiner la cogénération avec un système de stockage thermique dans le sol et une pompe à chaleur. Un tel système promet d'être plus flexible (augmentation du degré d'autosuffisance de 14,0 points de pourcentage) et plus efficace sur le plan énergétique (augmentation de 2,9 points de pourcentage) qu'un système conventionnel.

Les batteries seront un élément clé des futurs systèmes énergétiques - dans les véhicules électriques à batterie et comme dispositifs stationnaires de stockage de l'électricité. L'objectif du **sous-projet T3.6** était de mieux comprendre les facteurs qui contribuent à la dégradation des batteries et d'améliorer la longévité des batteries destinées aux applications stationnaires [26]. Pour ce faire, une campagne de vieillissement a été menée pour simuler des conditions réelles, y compris des scénarios standards et "abusifs". Quatre températures différentes, deux taux de charge/décharge différents et trois niveaux de profondeur de décharge (DOD) différents ont été testés. Divers paramètres ont été contrôlés au cours du processus de vieillissement, les principaux paramètres d'intérêt étant la perte d'efficacité par cycle et la perte de capacité de décharge. Ces deux paramètres sont utiles pour prédire la durée de vie de la cellule de la batterie. Les principales conclusions sont que la température a une forte influence sur les performances des trois technologies de cellules (NMC/LTO, NMC/Gr et LFP/Gr) étudiées. La DOD a eu un impact notable sur le processus de dégradation des trois technologies, en particulier pour les cycles de vieillissement à 0°C et 25°C. On peut également observer que (i) la technologie NMC/LTO est un choix supérieur pour les applications qui nécessitent un taux C standard, des performances stables, une plus longue durée de vie de la batterie et une plus grande capacité délivrée ; (ii) pour le cyclage à taux C élevé, le choix de la technologie et la température ambiante jouent un rôle critique dans la détermination des performances et de la stabilité à long terme de la batterie. Le NMC/LTO s'est avéré être le meilleur choix pour la stabilité à des températures élevées, tandis que le NMC/Gr a donné de meilleurs résultats à des températures intermédiaires de 25°C ; (iii) le vieillissement du calendrier et le cyclage à des températures élevées (50°C) ont entraîné une dégradation plus rapide pour les cellules LFP/Gr que pour les cellules NMC, ce qui fait du NMC un meilleur choix pour les applications à haute température.

Le **sous-projet T3.7** était complémentaire du **sous-projet T3.6**. Alors que les batteries permettent de stocker de grandes quantités d'énergie mais peuvent être facilement sollicitées par des charges ou des décharges importantes, les supercondensateurs peuvent fournir une puissance supérieure d'au moins un ordre de grandeur avec une durée de vie exceptionnelle de près de 100 000 cycles, mais leur capacité de stockage d'énergie est généralement limitée. Les supercondensateurs commerciaux de pointe reposent sur le mécanisme physique de stockage de la charge, appelé capacité à double couche électrique (EDLC). En raison de ces caractéristiques, l'utilisation de supercondensateurs peut être bénéfique pour atténuer les limitations des technologies de batteries actuelles liées à la lenteur de la charge et à la durée de vie limitée grâce à l'intégration en tandem. Pour réaliser cet avantage, les caractéristiques des supercondensateurs commerciaux doivent être bien comprises. C'est pourquoi le projet vise à (i) réaliser une évaluation complète du comportement électrochimique des supercondensateurs commerciaux, (ii) identifier les paramètres critiques pour maintenir une performance de cycle stable, (iii) et évaluer l'intégration potentielle des supercondensateurs dans les systèmes de stockage en réseau basés sur des batteries [27]. Les résultats ont montré que même avec des spécifications similaires, les cellules de supercondensateurs de différents fabricants peuvent se dégrader à des taux nettement différents, soulignant la nécessité de tests indépendants avant leur utilisation dans le stockage en réseau. L'un des résultats les plus remarquables est la variation de capacité d'environ 5 % d'une cellule à l'autre, qui peut entraîner une distribution inégale du courant et de la tension lorsque les cellules sont assemblées en modules, ce qui peut conduire à un fonctionnement dépassant les conditions recommandées par le fabricant. En outre, les cellules soumises à une tension légèrement supérieure à la tension recommandée se dégradent à un rythme nettement plus rapide. L'étude a également mis en évidence l'effet néfaste des courants élevés sur la conservation de la capacité, une situation qui s'aggrave avec l'augmentation de la température. L'imagerie thermique a montré que le fonctionnement à courant élevé entraîne des hausses substantielles de la température des cellules, ce qui souligne la nécessité d'une gestion thermique efficace dans la conception des modules. Enfin, l'intégration de modules batterie-supercondensateur a démontré qu'elle pouvait améliorer la durée de vie des batteries en minimisant les contraintes grâce à des plages d'état de charge plus étroites, ce qui constitue une voie prometteuse pour prolonger la longévité des systèmes de stockage d'énergie.

Toutes les fonctionnalités de la plateforme ReMaP ont été utilisées pour un total de quatre expériences HIL dans trois sous-projets. Un nombre encore plus important de projets a utilisé le cadre de simulation ReSIM. Les composants matériels et les capacités de contrôle améliorées qui ont été développés sur les deux plates-formes expérimentales ESI et ehub perdurent après la fermeture de ReMaP. En outre,

ReSIM constitue la base de l'analyse de la flexibilité des systèmes multi-énergétiques dans le cadre du projet SWEET PATHFNDR soutenu par l'OFEN.

Contents

List of Abbreviations

API	Application programming interface
ARE	Bundesamt für Raumentwicklung
BMS	Battery-management systems
CDF	Cumulative distribution function
CEA	City Energy Analyst
CHP	Combined heat and power
DSM	Demand-side management
Empa	Eidgenössische Materialprüfungsanstalt
ESI	Energy system integration (platform)
FFNN	Feed-Forward Neural Network
HIL	Hardware-in-the-loop
ICE	Internal combustion engine
LAV	Labor für Aerothermochemie und Verbrennungssysteme
LSAdam	Least-Square Adam
LSSGD	Least-Square Stochastic Gradient Descent
MES	Multi-energy systems
MILP	Mixed-integer linear programming
NSGA	Non-dominated Sorting Genetic Algorithm
OPC UA	Object linking and embedding for process control unified architecture
PAR	Passive-Aggressive Regression
PEM	Polymer electrolyte membrane
PSI	Paul Scherrer Institut
PV	Photovoltaic
QAdam	Quantile Adam
QPAR	Quantile Passive-Aggressive Regression
QSGD	Quantile Stochastic Gradient Descent
REST	Representational state transfer
SAIFI	System average interruption frequency index
SAIDI	System average interruption duration index
SMR	Steam methane reforming
TES	Thermal energy storage
UBEM	Urban Building Energy Models
VEP	Venios Energy Platform

1 Introduction

1.1 Background

The integration of high shares of fluctuating renewable energy sources and the transformation of the classical role of consumers to “prosumers” are, amongst others, significant challenges for the energy systems of the future. Especially in distribution grids, where the usual energy flow is directed from the transmission network towards the consumers, significant impacts like overloading of grid components might occur. To forestall or delay the need for grid infrastructure upgrades and to improve system safety, stability, and power quality, a thorough understanding of the energy system of the future is crucial. There is thus a clear need for a research platform that allows testing the combination and interaction of energy technologies and carriers; the interaction between the distribution system, buildings, and mobility; and to educate students and experts in renewable energy technologies and decentralized energy systems in a close-to-reality environment.

The multidisciplinary flagship project “Renewable Management and Real-Time Control Platform” (ReMaP) aimed at meeting the above-mentioned need by developing a flexible, hardware- and software-based research platform for assessing potential energy-system solutions for the neighborhood of the future. ReMaP enabled the testing, analysis, and optimization of multi-component, multi-energy carrier systems on the distribution level, fostering the collaboration of multi-disciplinary research teams from both academia and industry, and provided a control and communication infrastructure for the joint operation of existing platforms and demonstrator sites.

The core of the project was the design, building, and deployment of the modular ReMaP platform, which includes the setup of a communication and real-time data and control system that is connected to the Energy System Integration (ESI) platform at PSI and the NEST, move, and ehub platforms at Empa. ESI is a research and technology-transfer platform on which novel larger-scale solutions for energy conversion and storage can be developed, up-scaled, evaluated and validated, whereas NEST, move, and ehub are demonstration platforms for innovative building technologies, future mobility solutions, and energy management up to the district scale, respectively.

1.2 Goal and objectives

The overarching goal of the ReMaP project was to design, build, and deploy an advanced technology platform of national and international relevance that will contribute to the understanding of future energy supply systems. The specific objectives of the project are:

- To enable academic and industrial researchers to assess the performance of energy conversion and storage technologies that are embedded within smart grids with advanced real-time measurement and control capabilities.
- To provide instructors with a unique tool to educate the next generation of scientists, engineers, and technicians that is needed to deliver the energy system of the future.
- To demonstrate the practical use and benefits of energy conversion and storage technologies in smart grids to the relevant societal actors and stakeholders, especially the municipal, cantonal, and federal authorities as well as the general public.

1.3 Structure of this report

The present document mainly describes the ReMaP platform, i.e. an ensemble of software elements that were created during the project to allow for the simulation of future multi-energy systems with focus on the issues related to controlling such systems. Here simulation may be solely on a computer or including real-world hardware elements (HIL – hardware in the loop), for instance at the ESI platform of PSI or the ehub platform of Empa. The part of the software that interfaces with the real world – the control framework – was created in collaboration between the project partner SGS and the software developers at SCS. It is described in Section 2.1 and in two annexes (Sections 7.1 and 7.2). The simulation framework – termed ReSIM – is described in Section 2.3. The software is further documented in two annexes to this report (Sections 7.3 to 7.4).

The second pillar of the ReMaP project was a number of subprojects that studied in detail specific elements of future energy systems. These projects were related in various ways with the ReMaP platform (see Section 4.10). Some delivered new software elements (See Section 2.3.3), others used the full HIL functionality of the ReMaP platform (see Section 3). Each of these projects is described with its own report that can be found on the ARAMIS platform[1]. Section 4 of the present report gives a short summary of the major results.

Subproject	**Topic**
T3.1	Distributed State Estimation and System Operation
T3.2	Probabilistic Load Prediction for Optimized Grid Operation Capacitive Energy Storage
T3.3	Design and operation of reliable, decentralized multi-energy systems
T3.4	Virtual Neighborhood Load Emulation
T3.5	Improved combined heat & power concepts
T3.6	Battery storage
T3.7	Capacitive energy storage
T3.9	Advanced control algorithms for future energy systems
T3.10	Electrochemical energy storage

[1] https://www.aramis.admin.ch/Grunddaten/?ProjectID=41788

2 The ReMaP Platform

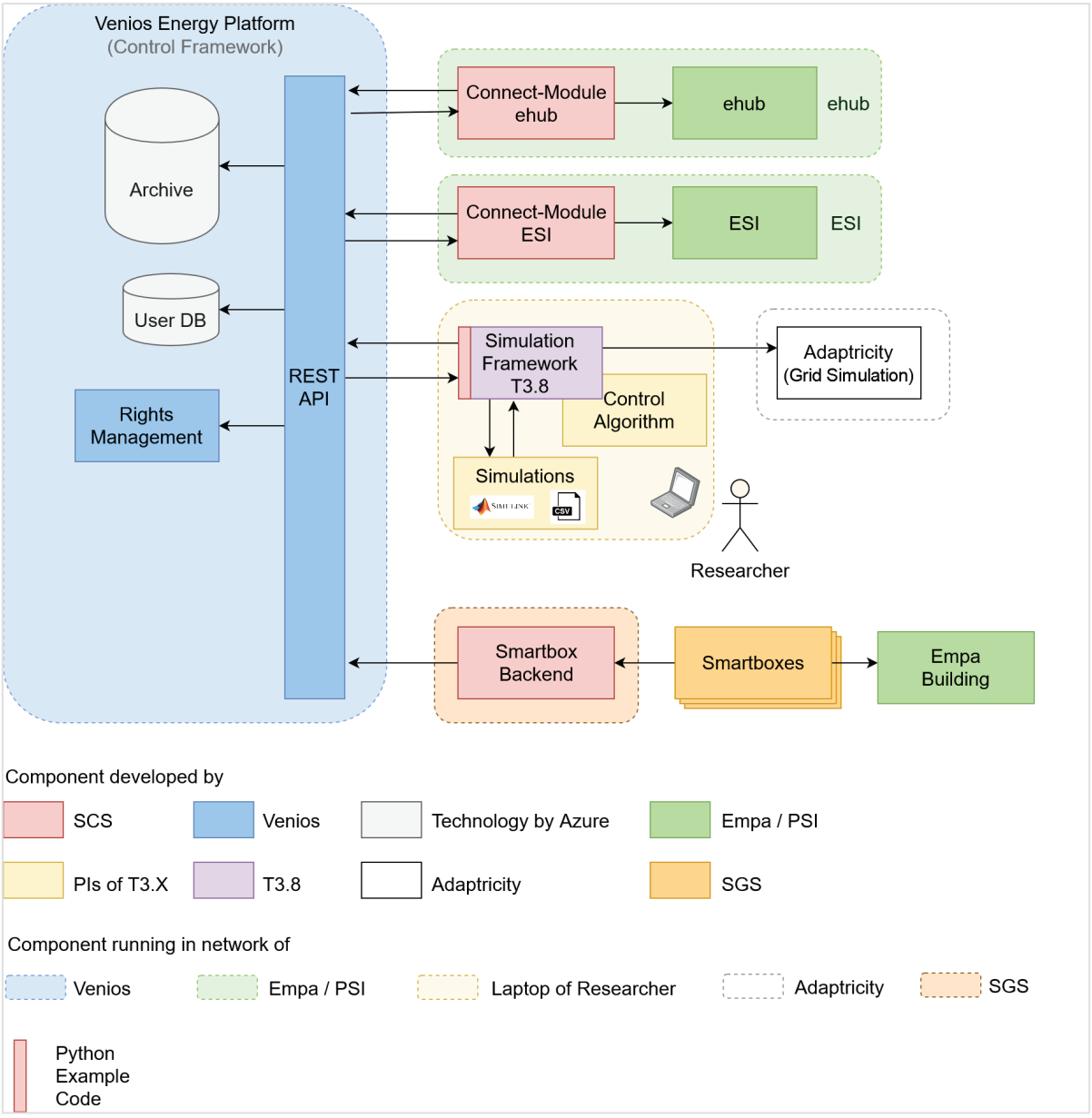


Figure 1: Architecture of the ReMaP Platform.

The ReMaP Platform is an ensemble of software and hardware that was developed during the project. It allows to study future energy systems from a pure simulation on a computer to an experiment that combines physical hardware at different locations. It especially connects the two main experimental facilities ehub (Empa) and ESI (PSI). Its basic architecture is shown in Figure 1.

The two main components of the ReMaP platform are the Control framework and the Simulation Framework. The first was developed by the project partner SGS in close collaboration with Supercomputing Systems (SCS). It is based on the Venios Energy Platform (VEP), hosted on Azure

(see left side of Figure 1). It serves as the interface to read- and control-values of hardware and access to archived measurements. ehub (Empa) and ESI (PSI) are connected to the VEP in an almost identical way: Each has its own ConnectModule, running in the ehub/ESI infrastructure, as a Windows service.

To create an experiment, simulations and control algorithms have to be integrated in the Simulation Framework, that was developed within Subproject T3.8 and was recently labelled ReSIM (ReMaP SIMulation Framework). ReSIM can then be run on a standard PC or laptop. It connects to VEP with the provided helper functions (Python example code, see Section 2.1.5). ReSIM and the Python example code serve as building blocks to write the code for a new experiment. The Annex to this report contains specifications and operation manuals for both the control and the simulation framework.

2.1 Control framework

2.1.1 Datamodel

To integrate ehub, ESI and ReSIM, it was crucial to decide on a good strategy for the data model. On the one hand, it should be easy to run the same control logic in different setups (with different hardware components and different simulations). Therefore, there is a need for a unified data model. On the other side, if we compare for instance three batteries (one at Empa, one at PSI and one simulated), the components have different sensors and different control values. Even more, hardware at Empa and PSI is grouped differently and the hierarchies in their systems do not map to each other.

Therefore, the data model was chosen in the simplest way on basis of a signal. Each signal has a VeniosId (e.g., ehub.65NT.EXP20.T200.ElectricalStorage.BidirectionalInverter.ActivePowerTotal) and a type (e. g. float). The VeniosId gives a hierarchy on where the signal is located. By this, signals can still be grouped (by the party providing the signal), but the user can combine them in the way he needs. This makes the end user (using existing signals in a control algorithm, for example) independent of the party providing the signal (e. g. PSI).

2.1.2 Writing Control-Values

One advantage of using a common platform for ehub and ESI is, that the protocol to send control values, is the same for both systems. To get control access (and to keep it), set_research_mode_on needs to be send continuously. This is needed to be able to send new control values, but also so that the last set value is kept and not changed to a fallback value from the hardware itself (see Figure 2). The differences in gaining control access in ehub and ESI are handled by the corresponding ConnectModule and not displayed here.

2.1.3 Rights Management

One of the requirements of ReMaP is, that a user should only have access to data, of specific devices, from a specific time frame. This was done by modelling an "experiment" in Venios (see Figure 3). An experiment has a time interval (start, end), a list of devices and a list of users (within different usergroups). The users can access any measurements from the devices, that have a timestamp within the time interval (also when the timestamp is in the past). Control access to control-devices is only given within the time interval. So once the time interval is completely in the past, no user can control the devices of the experiment anymore.

2.1.4 ConnectModules

The hardware at Empa and PSI is both integrated in Venios with their corresponding ConnectModules. The ConnectModules are written in C# and run as Windows Services in the infrastructure of the corresponding institute. Both have their own configuration file for connected signals. So to integrate new hardware from ehub/ESI, only this configuration-file needs to be adjusted. It is also possible to run several ConnectModules in parallel, with different signals connected. This was done e. g. at PSI, to integrate Cosyma, which is located outside the PSI campus, into ReMaP.

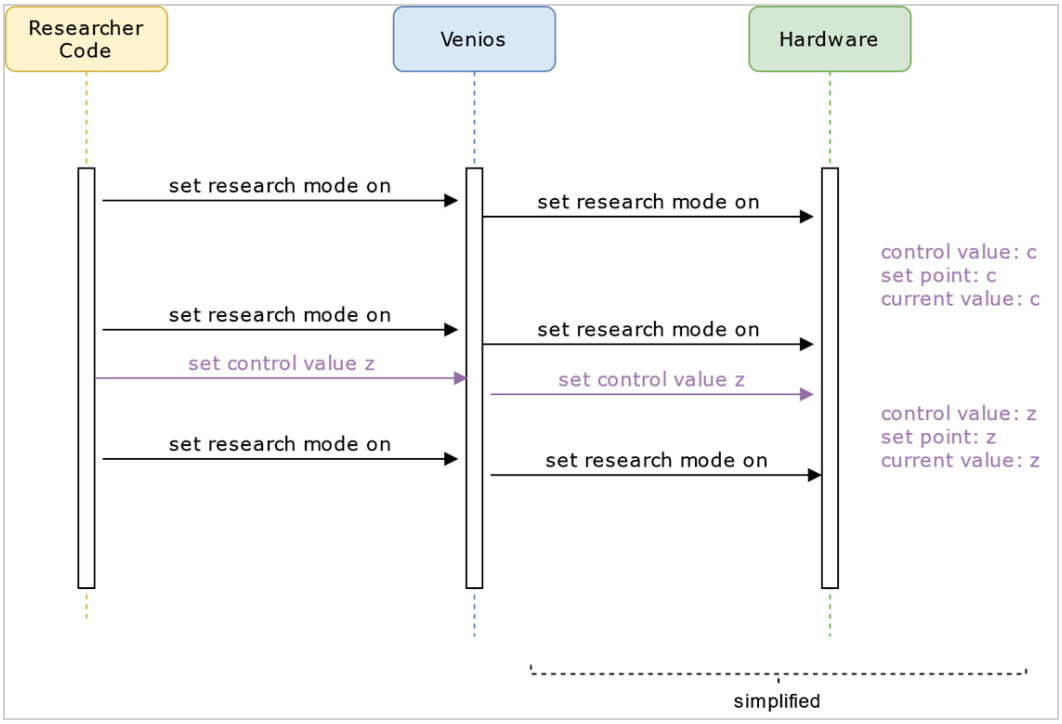


Figure 2: Sequence diagram on writing control values

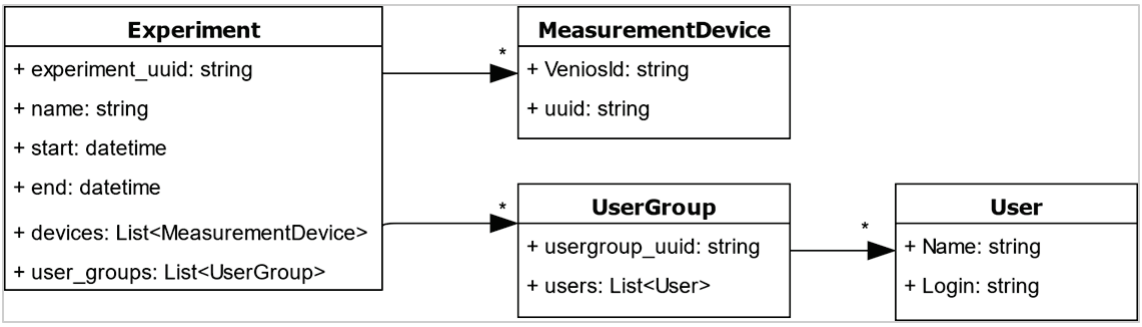


Figure 3: Class diagram on Experiment.

2.1.5 Python Example Code

Python is a programming language commonly used in research. Therefore, example code was provided in Python, to show how to interact with VEP. Not only does it provide examples, but it contains functions and classes, that simplify the interaction with VEP and cover basic use cases, like checking the connection to the hardware at program start or keeping control access throughout the experiment. To use it, the researcher needs to define a signal-list. The defined signals can then be used from within the researcher code to interact with VEP. A more detailed description is found in the Annex (Section 7.2).

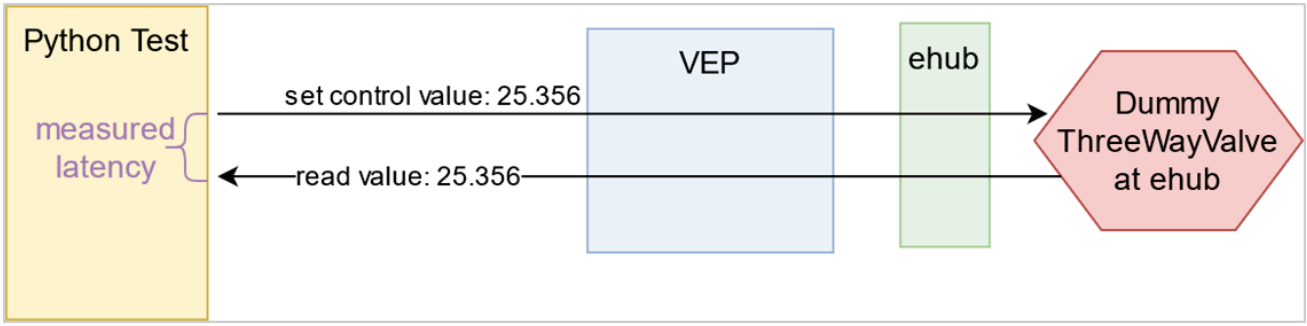


Figure 4: Process for measuring latency

2.1.6 Security

Depending on the data, there are certain requirements on storing and accessing the data in ReMaP. Particularly PSI has strict security requirements for the data access in ReMaP. These are fulfilled with the current version of the ReMaP system. The following measures were particularly implemented to conform with the regulatory requirements of PSI:

- VEP runs on Azure Switzerland
- User access is restricted to the signals of an experiment
- Communication between ESI and VEP is always started/opened from within the ESI network, i. e. the ESI network is closed to the outside.

2.1.7 System Performance

The system has a natural limit to its performance since it connects hardware at different locations over the internet. Furthermore, there are only few experiments, that need a latency in the subsecond range. Therefore, the goal was set to implement a system that can guarantee a latency of 1 sec. Meaning it should be possible to react to a changing hardware signal and set a new control value within 1 second. As this scenario is hard to measure (as it requires synced timestamps at the hardware and the Python Code), it was decided to measure the time between sending a control value to VEP and reading the same value from the corresponding read-signal (see Figure 4).

The results show, that the average latency currently depends on the amount of read-signals connected to VEP. The measurements were taken in a period of 2 hours, where the control value was continuously changed, every couple of seconds.

# Connected signals	Average latency for control loop
4	1.6 s
50	1.9 s
100	2.3 s

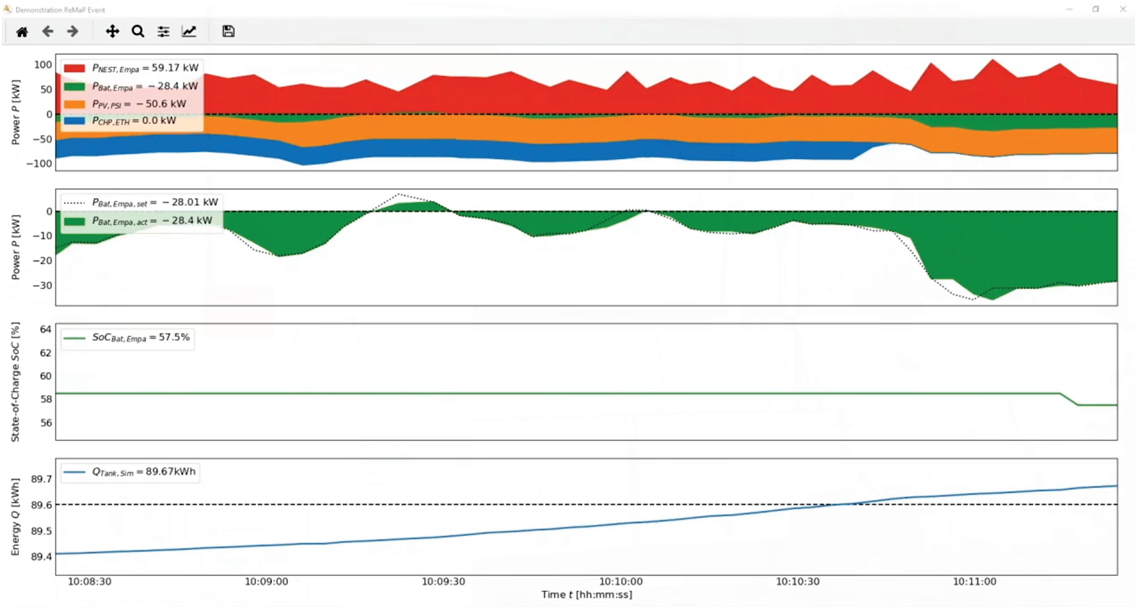


Figure 5: Visualization implemented by Empa for the ReMaP Event in December 2020

2.1.8 User Interface

A simple visualization was implemented in Python to show live data of experiments (see Figure 5).

2.1.9 Example: Interaction of Components in Demo-Use case

The following gives an illustrative example on the flow of information through the whole ReMaP system. The use case is: A control algorithm continuously checks the PV production at PSI. Depending on the value, the setpoint for the energy production of a battery at NEST is set (see Figure 6).

1. Before the control logic starts, the communication with the ReMaP system has to be setup, this includes:
    a. Checking if all read-signals have current values
    b. Requesting and waiting for control access to the battery at NEST
2. The PV production is continuously persisted in VEP
3. The control algorithm continuously checks the most recent value of the PV production
4. The control algorithm decides, if a new control value should be send to the battery at NEST
5. A new control value is sent to VEP. The ehub-ConnectModule receives the new control value over a WebSocket connection from VEP and forwards it to the OPC UA Server of ehub. The ehub OPC UA Server forwards the control value to the battery.
6. The control algorithm continuously sends a "set research mode on"-command to VEP. This triggers the ehub-ConnectModule to send the corresponding watchdog-commands to ehub. This way the researcher stays in control of the battery throughout the experiment.

Rights management: VEP makes sure that researcher and the ConnectModules have the corresponding rights needed. For example, the PV signal from PSI can only be archived by the ESI-ConnectModule, but not by the researcher and also not by the ehub-ConnectModule.

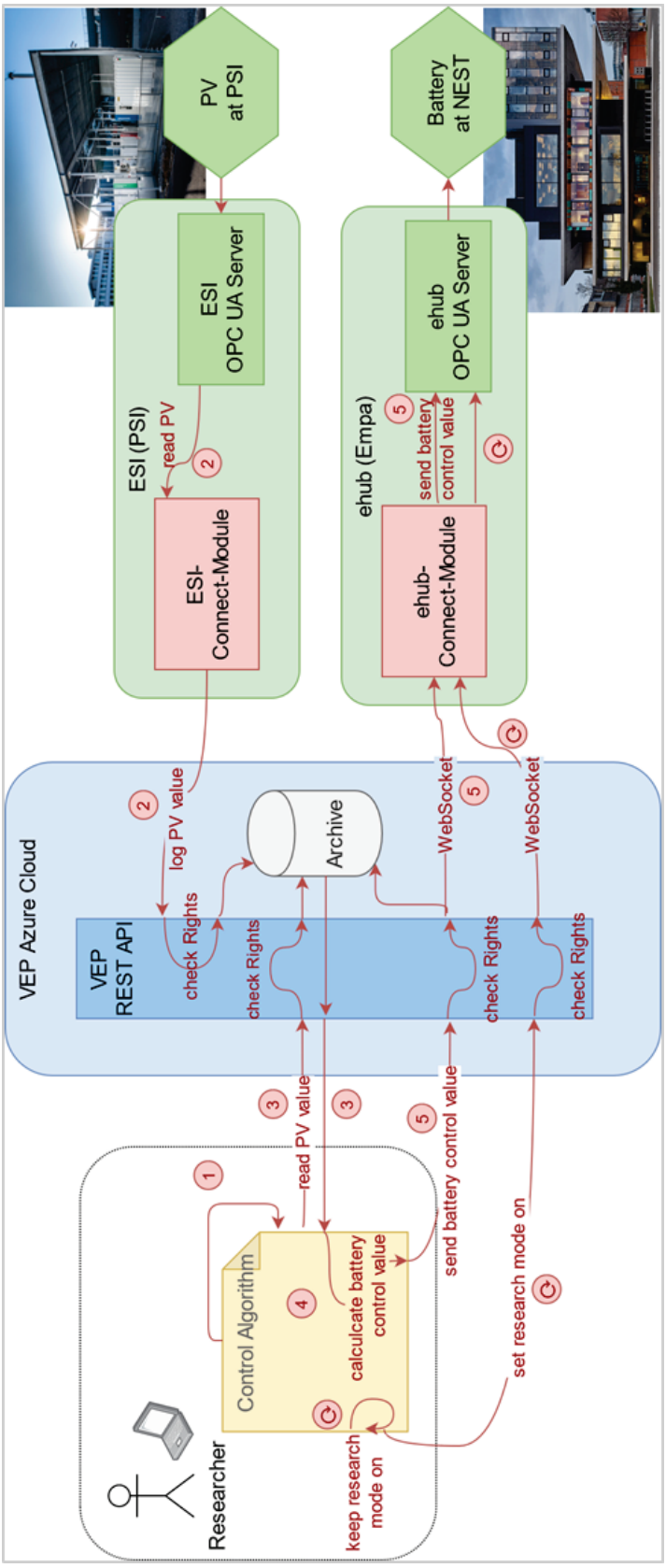


Figure 6: Data flow for a simple experiment

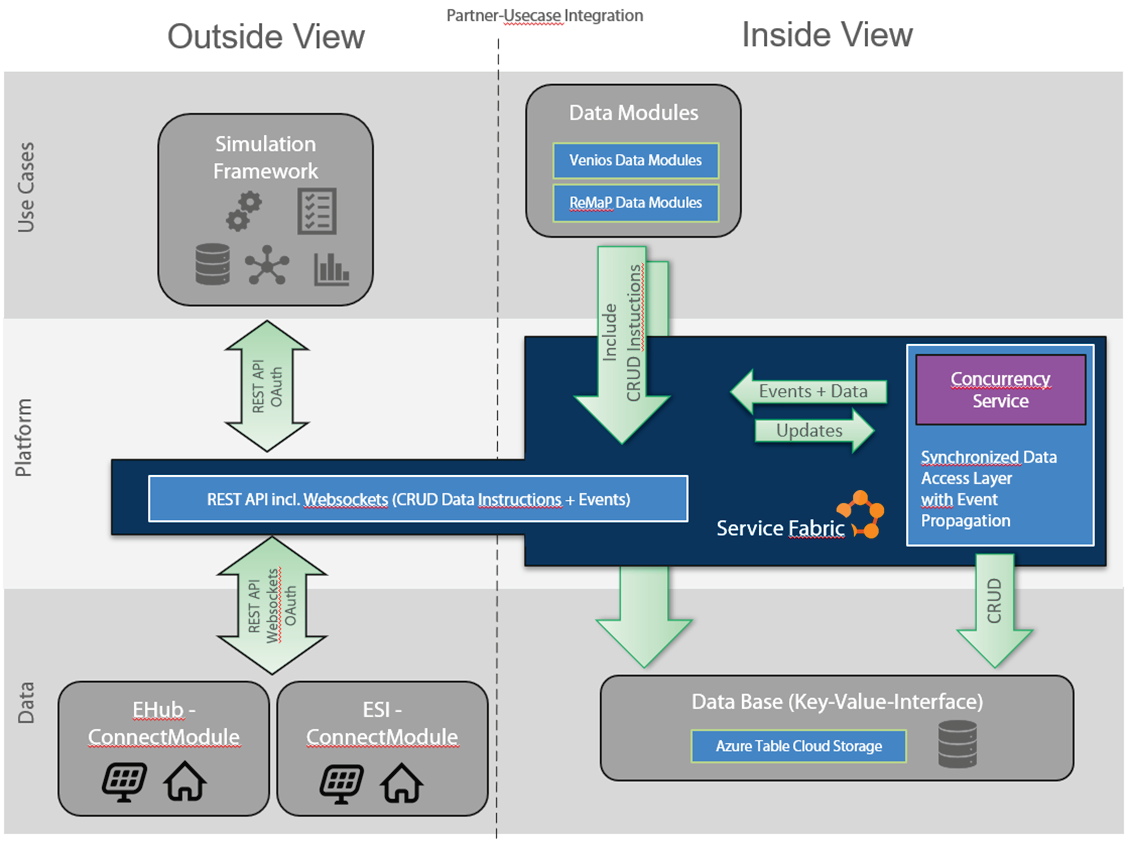


Figure 7: Overview on Venios Energy Platform and its Integration in ReMaP

2.2 Venios Energy Platform

The Venios Energy Platform (VEP) has passed through multiple iterations during the work on the ReMaP project (see Figure 7). While the project started on the basis of VEP 3, the development of VEP 4 has already started in autumn 2020. This version incorporated many improvements, which have been identified as essential during the initial ReMaP experiments.

The system is natively designed as a distributed system and allows for easy out-scaling in case of an increase in requests. There are basically no restrictions on the amount and frequency of data to be ingested. Central rights management is controlled for each request to the system and can be configured on an entity level. The entire system supports the development of reactive applications. This means that it is possible to be informed about changes in the system according to a classic publish-subscribe pattern. In addition to access via the RestAPI, events can be actively sent to clients of the system via websockets.

By designing the platform as a microservice architecture, it is possible to add further services to the platform in the future. Communication within the platform also takes place via websockets. There is a central lock service that orchestrates the edits of the system.

Hosting of the platform is done in the Microsoft Azure Service Fabric. VEP 4 was written in .NET 5/6. Azure Table Storage is used as the database. In principle, however, it is possible to use any database with a key-value interface. This ensures that there is no long-term lock-in against Microsoft.

During development, strict attention has been paid to separating ReMaP-specific adjustments from the general functions of VEP 4. All basic platform functions are developed together with the basic base classes of the database schema for all instances of VEP. This includes the data access layer, rights

management, data structures such as timelines, the logging system, the message system, and the provision of a basic CRUD API for access. For projects like ReMaP, the data model is compiled based on the given data structures and extended for the specific use case. Furthermore, a high-level API specially adapted for ReMaP is provided which abstracts the basic CRUD commands and is adapted to the data classes of ReMaP. By using the same platform kernel in all projects with VEP, it is possible to guarantee improvements and updates with a high frequency.

The core of the platform is used in many industrial projects both in the DACH area as well as internationally. This involves more than 40 active instances at German DSOs related to the German Redispatch 2.0 regulation. The platform is also used in projects in Switzerland at Elektrizitätswerke Zürich, in the Greencity project, and in the field of TSO/DSO coordination together with Swissgrid. The design of the platform allows easy communication between different instances of VEP if required.

2.3 ReSIM – ReMaP Simulation Framework

2.3.1 Basic setup

ReSIM, the Simulation Framework developed during the ReMaP project, is a software framework allowing to simulate the operation of all energy sectors in a district. It has the following characteristics:

- It models both utility-level and site-level components.
- It allows for simultaneous representation of a plurality of operational logic / controllers.
- The time step and simulation horizon are selected by the user depending on the considered case.
- It can be used as part of a hardware-in-the-loop simulation with devices integrated in the ReMaP platform via the control framework.

Figure 1 illustrates an example of what can be modeled in ReSIM: utility- and site-level infrastructure and components, as well as a plurality of controllers. ReSIM emulates how such a system would behave.

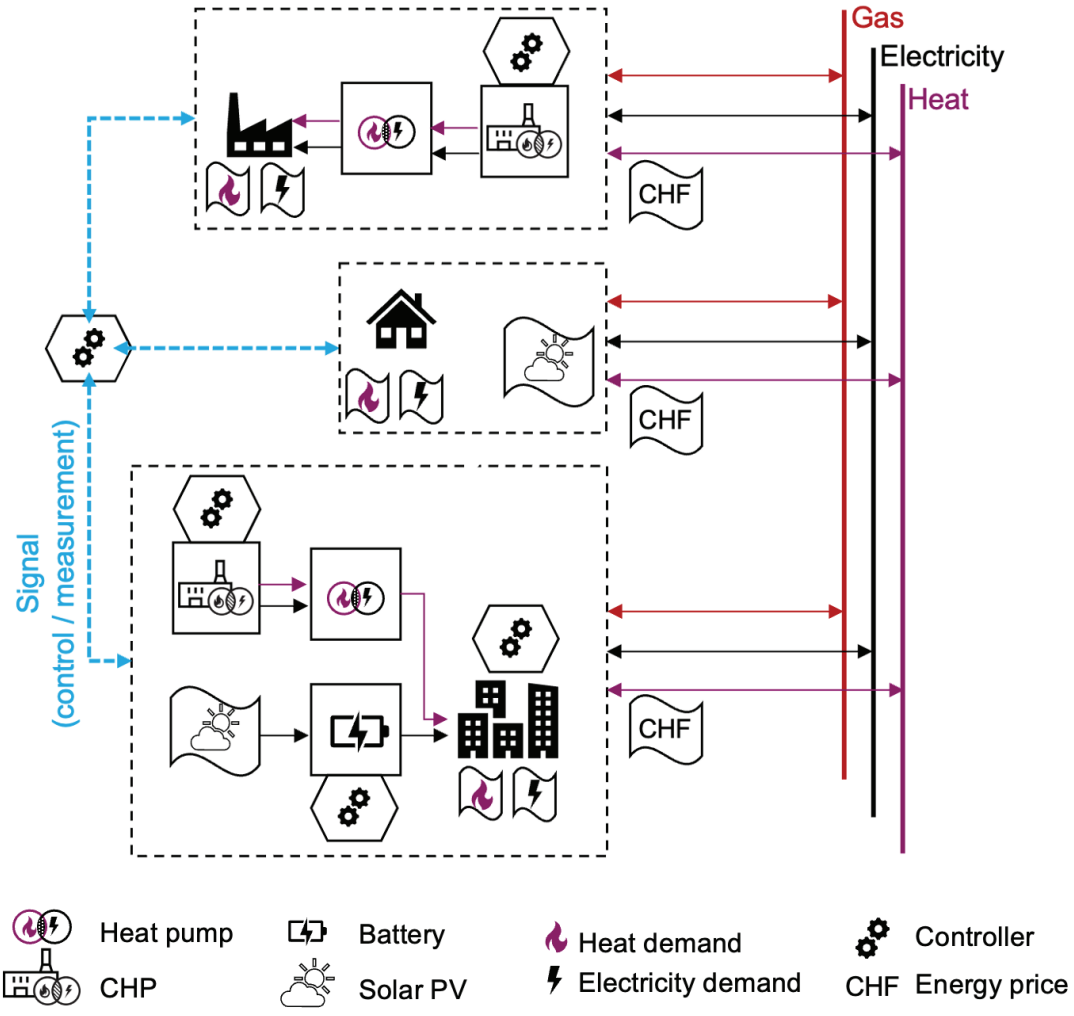


Figure 8. Example of a system modelled in ReSIM

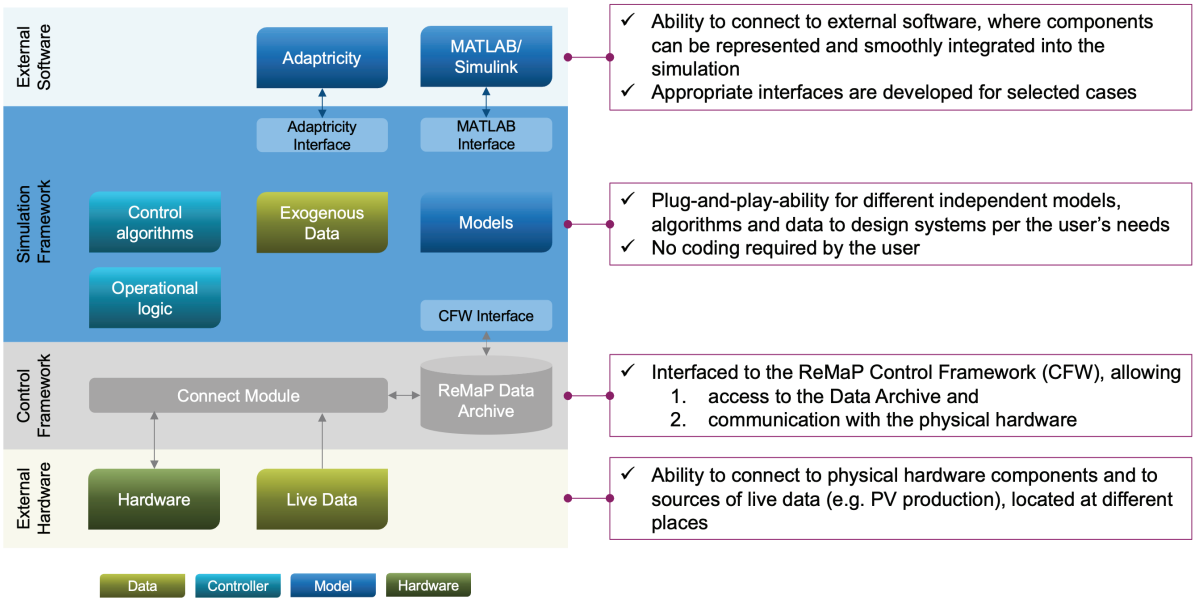


Figure 9. ReSIM (Simulation Framework): Modular structure and connections to software and hardware

Figure 9 illustrates how ReSIM interacts with the ReMaP Control Framework to eventually read Live Data or to perform Hardware-in-the-Loop simulations. It also illustrates the modular logic of ReSIM (where each model or algorithm/logic is a module) and the fact that it can interact with external software. ReSIM has been successfully utilized in several ReMaP subprojects, some of them pure simulations (T3.1, T3.2, T3.3), other including physical hardware (T3.5, T3.9 and T3.10, see also Section 3). Figure 9 positions the Simulation Framework in the context of the entire ReMaP platform as well as its interactions with software, hardware and sources of data which are not part of ReSIM.

The main features of ReSIM are as follows:

- Plug-and-play-ability for different independent models, algorithms and data, which one or more users can provide and aggregate together, eventually making up systems to be simulated. This is an important feature of ReSIM, as it allows for different users to independently develop their models, algorithms or even entire systems, each using his/her own tools and software, and then integrate them together in a smooth-less manner and simulate the combined system.
- Each independent hardware model and control algorithm is added to a "Model Library", which is continuously expanded as more users are utilizing ReSIM. It contains models, data and operational algorithms, which a user can use to build his/her simulated systems. The more ReSIM is used, the richer its model (and algorithm) library becomes.
- The design is modular; a user can add his/her own models or algorithms. There is no pre-defined required level of complexity per model or control logic/algorithm. The user selects the required level of modelling detail and algorithm complexity. In all cases, the required models and algorithms need to be provided as modules in the Model Library.
- Ability to connect to external software, where components can be represented and then smoothly integrated into the simulation. So far, two such software have been integrated: the Adaptricity power flow solver (see Section 2.3.2), where communication with ReSIM takes place via a rest API and file exchange, and MATLAB/Simulink, where communication with ReSIM takes place utilizing the Matlab Engine module of Python. This feature significantly enhances the plug-and-play-ability feature described above.

- Ability to connect, via the Control Framework as illustrated in Figure 9, to physical hardware components and to sources of live data (e.g. PV production), located at different places. Communication with ReSIM takes place via the Control Framework. Eventually, ReSIM reads, in real-time, measurements and data from the ReMaP Data Archive and write, again in real-time, actuation commands to the Data Archive.

The main software design attributes are as follows:

- The core code is written in Python.
- Systems are represented as a nested hierarchy of systems, where each system consist of a set of sub-systems, a set of physical components, exogenous data sources and an operation/control logic, as schematically illustrated in Figure 10.
- An object-oriented approach is followed, where models and algorithms make up a hierarchy of classes.
- A simulated system is described in a text-based format, by a set of json files.

A documentation of the ReSIM software is found in the Annex, Sections 7.3 and 7.4.

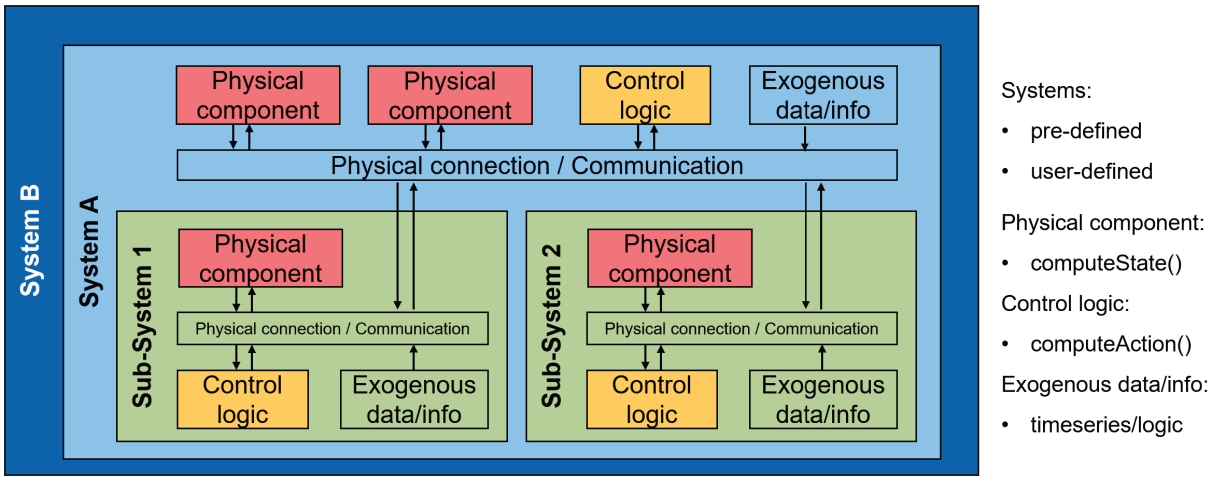


Figure 10: Definition of a system in ReSIM as an hierarchy of nested sub-systems.

2.3.2 Interface to grid model by Adaptricity

Adaptricity AG is a spin-off company of ETH Zurich. It was founded in March 2014, and renamed to Secure Switzerland AG in May 2022. The company develops, sells, and operates “Adaptricity”, a cloud-based platform for the simulation and analysis of power distribution grids. Adaptricity allows distribution network operators to better understand, manage and plan their grid infrastructure using data-driven network analysis. For simplicity we refer to the tool and the company with the common name Adaptricity.

Adaptricity constitutes the engine for grid-modelling and simulation of ReSIM. Within ReMaP, we developed REST-APIs for the integration of the engine with ReSIM. The most relevant APIs allow to:

- automatically upload of grid-models.
- run power-flow simulation.
- retrieve and export results for further processing.

Adaptricity modelled the full low voltage distribution grid of the three demonstrators at Empa Dübendorf, i.e. the NEST building, move and ehub, and performed a time-series simulation with 5-minute resolution

measurement data for the year 2019 (see Figure 11). Similarly, Adaptricity modelled the medium and low voltage grid at PSI, however no simulations were possible yet due to unavailable measurement data. Within ReMaP. the graphical grid editor turned out to be an important feature of Adaptricity, enabling the researchers to quickly model and edit the grids.

Figure 11: Grid model at Empa.

General support for grid modelling

Adaptricity provided support for modelling and troubleshooting of grid models. Consulting helped ensure that the power grid model was accurate, reliable, and reflective of real-world conditions. The working group involved consultants who had expertise in power system engineering, data analysis, and software development. Adaptricity provided guidance on data collection, model formulation, and simulation techniques, and helped identifying potential limitations and uncertainties in the model.

Provision of synthetic grids

Within ReMaP, Adaptricity collaborated with City Energy Analyst and provided a set of synthetic grid models for the electrical modelling of small neighbourhoods. The two sample areas were placed in Zug and Zurich Altstetten (see Figure 12).

During the project, researchers suggested to integrate in Adaptricity a data-converter to export the grid models in MATPOWER format. The converter was developed and made available during 2021. Thanks to this option, it was possible to leverage Adaptricity's graphical grid editor to model the study-grids and then export the grid-models to MATPOWER for further processing. Specifically, researchers had the possibility to quickly generate the admittance matrix of the grids, which is a required parameter for control algorithms.

Figure 12: Synthetic grids for two sample areas in Zug (left) and Zurich Altstetten.

2.3.3 Further additions to ReSIM from sub-projects

Various subprojects have provided software that could be added to the ReSIM model library. This section gives an overview.

Online feedback optimization (OFO) controller (Subproject T3.1, Section 4.1)

Within Subproject T3.1 a novel Online Feedback Optimization (OFO) controller was developed to operate grid based on real time grid-state measurements and applied to representative test cases [1]. This OFO approach was added to the ReSIM model library as a controller named “OFOController”, that includes the functionalities of methods 1 and 2, that is, OFO with quadratic operator and model input-output sensitivity learning.

Probabilistic load forecasting algorithms (Subproject T3.2, Section 4.2)

Within Subproject T3.2, a probabilistic load forecasting algorithm was developed that combines online and offline learning. Historical data were used to train multiple regression models, generating independent forecasts. These models were then combined into an ensemble model for enhanced accuracy. Both deterministic and probabilistic models were applied to both building and individual appliance levels. Results showed that deterministic models are less effective due to load volatility, while probabilistic models provide improved uncertainty intervals. This approach was integrated into the ReMaP Simulation Framework (SFW) to predict building-level power flows [6].

City Energy Analyst - CEA (Subproject T3.4, Section 4.4)

Within Subproject T3.4, a connection between the CEA and ReSIM was established [21]. Here, the exogenous data, leveraging the energy demand outputs from the current sub-task and additional outputs from the solar panel simulation tool in the CEA were used to quantify decentralized electricity production potential of photovoltaic panels on buildings. The resulting time-series electricity production and consumption data were then connected to the grid simulation software by Adaptricity (see Section 2.3.2).

3 HIL experiments on ReMaP Platform

The full functionality of the ReMaP Platform was used in a total of four hardware-in-the-loop (HIL) experiments, i.e. combinations of simulations on ReSIM with real hardware that is connected via the control framework. A detailed description of the experiments is given in the respective reports of the subprojects [25][30][39]. This section gives a quick overview on the setup and results.

3.1 Optimal operation of CHP plant (Subproject T3.5)

Within Subproject T3.5, options to improve the flexibility and exergy efficiency of a combined heat and power (CHP) system were investigated. The most promising options were further tested with hardware-in-the-loop (HIL) experiments on the ReMaP platform. Figure 13 shows the setup for the first option, a real CHP plant combined with a simulated thermal energy storage (TES). Figure 14 illustrates the results of this experiment in comparison with a pure simulation in ReSIM. It can be seen that generally there is a good agreement. The details can be found in [25].

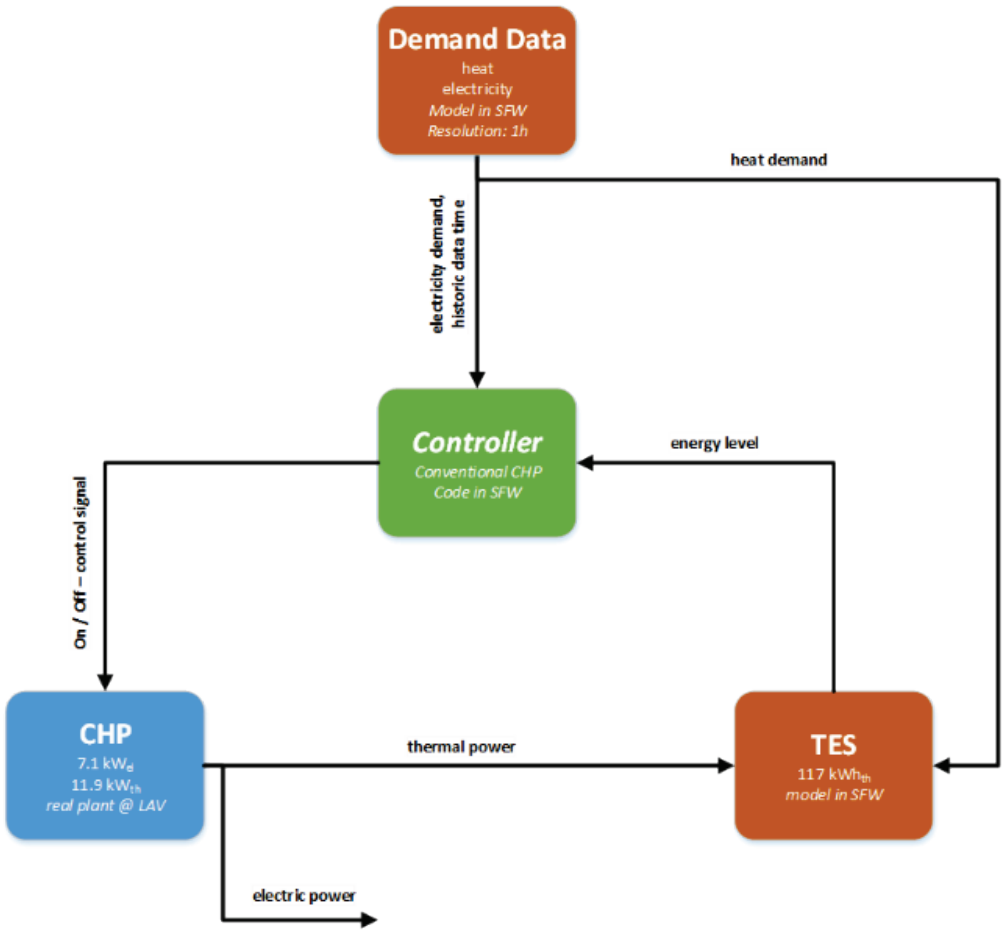


Figure 13: HIL layout 1: using CHP hardware along with hourly resolved demand data and TES model integrated in the SFW

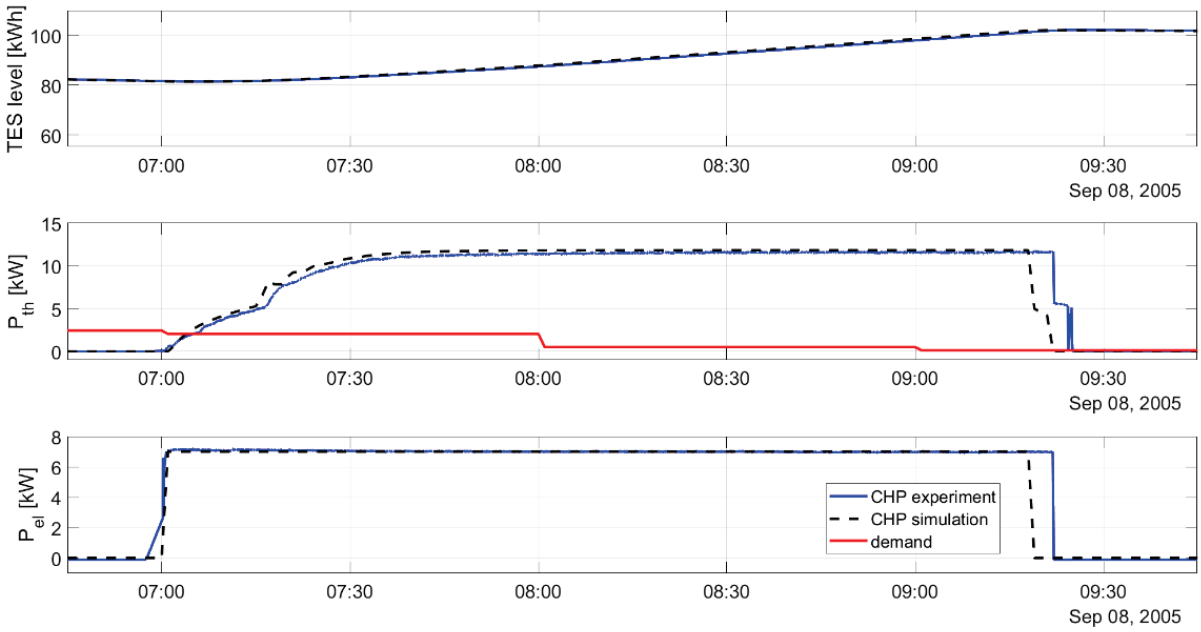


Figure 14: Results of the first HIL experiment. Top plot: TES level over time. Middle: Thermal power output as well as heat demand (red). Bottom: Electrical power output. Simulation results are depicted with black dotted lines while HIL results are displayed in blue.

3.2 Peak shaving algorithm (Subproject T3.9)

Within Subproject 3.9, several new control strategies for problems like optimal battery electric vehicle (BEV) charging were developed. One specific subtask considered the problem of mitigating the stress on the power grid by minimizing the daily power peaks of EV charging stations using hydrogen-based storage facilities. A new control strategy was tested with HIL experiments involving ReSIM and the ESI platform at PSI. Figure 15 gives an overview of the control architecture, details can be found in [30].

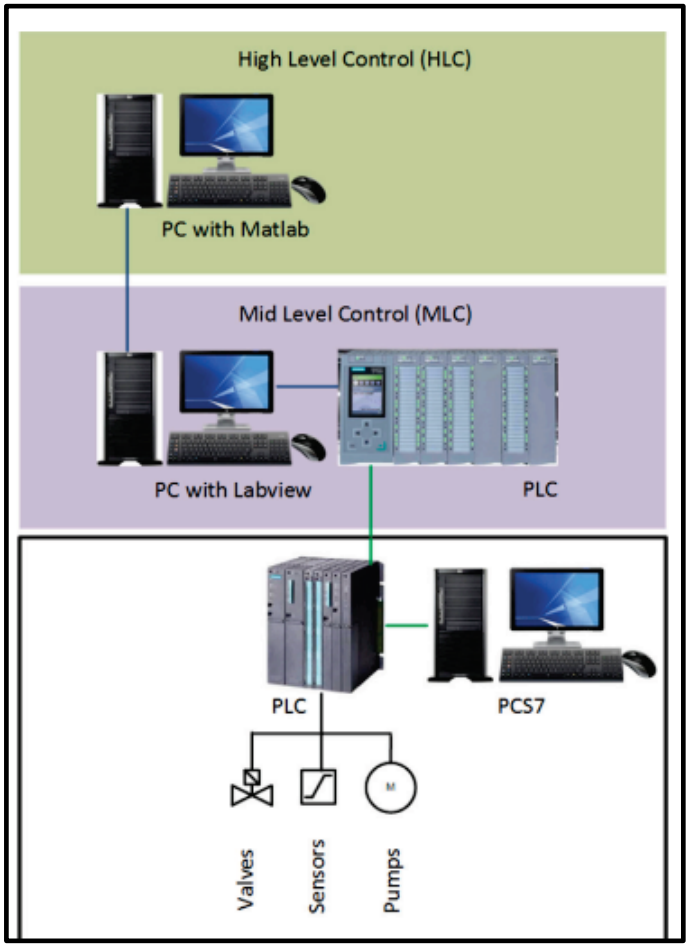


Figure 15: Control architecture showing the interconnection between HLC (running the MPC), the MLC and the LLC

The experimental results are shown in Figure 16. There is a good agreement between the real system behaviour (green line, ESI experiment) and the algorithmic solution (purple line, CCP+HESS), leading to the conclusion that the derived models are highly accurate. Note that after 18 hours there was a system shutdown, followed by a restart with a different initial condition (ESI experiment – restarted). The results are therefore not representative from that point on and are omitted from the analysis.

Most importantly, the daily peak achieved with the proposed algorithm is 288 kW in contrast to the 300 kW required by a strategy with storage but without smart charging (blue line, NCP+HESS) and the 350 kW required by a strategy that uses neither storage, nor smart charging (red line, NCP).

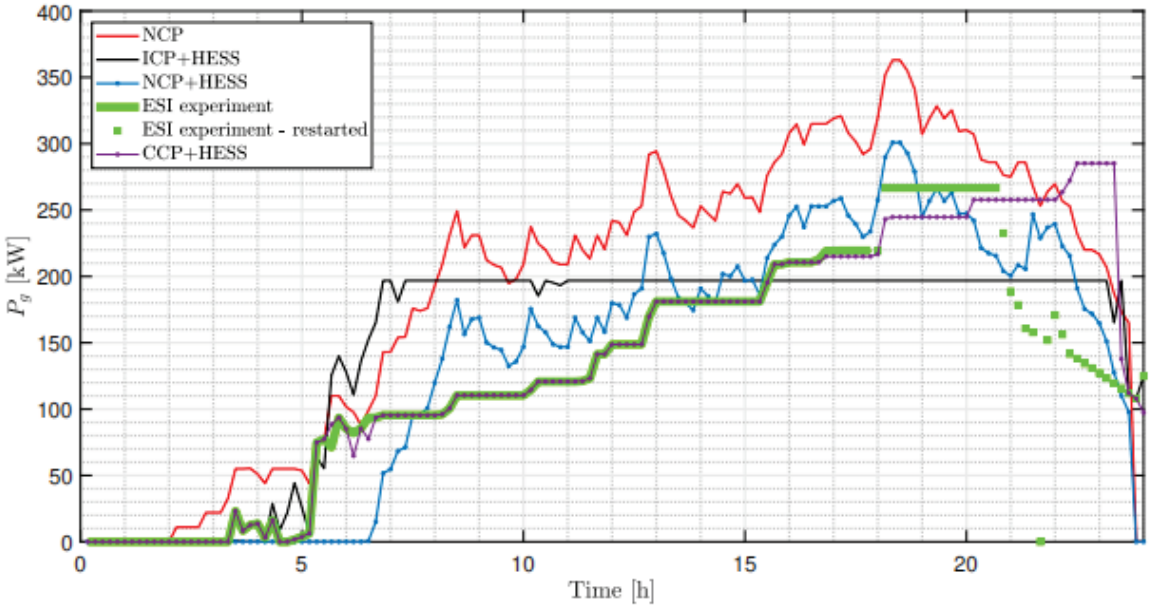


Figure 16: Experimental result for the peak shaving algorithm over one day at ESI.

3.3 Uninterrupted hydrogen supply for mobility (Subproject T3.10)

Within Subproject T3.10, another problem related to mobility was studied, namely hydrogen for fuel cell vehicles [39]. The goal of this experiment was to demonstrate how a constant supply of hydrogen for a mobility application can be realized based on high share of photovoltaics. The system consisted of an electrolyzer, a short-term hydrogen tank, a methanation reactor, a methane tank and a reformer. Figure 17 shows the setup for the experiment.

From the modelling/simulation, the different sizes of components and their operation schemes were known. The technical most challenging operation patterns (fast gradients, low part load of both electrolysis and methanation) were tested on the ESI platform using the electrolysis, gas cleaning & drying, and H_2-tank at PSI, the container based methanation unit connected to a real biogas plant in Inwil as well as the battery at Empa, all connected via the ReMaP platform.

Five days from the simulation of a complete year were chosen, during which the capacity of the methanation unit was varied from 0% to 100%. As the methanation unit in reality would be operated in a relatively stable pattern (using the hydrogen tank as dampening element, it was possible to scale the experiment by a factor 24 with respect to time. It could be shown that the methanation plant is able to handle part load changes without compromising the gas quality (see Figure 18). More than 96% methane content and less than 4% of CO_2 could be reached at any time. Only the hydrogen concentration peaked twice above the limit of 2% which is today is not needed anymore according to the recent regulations, but still important due to the fueling stations for CNG cars.

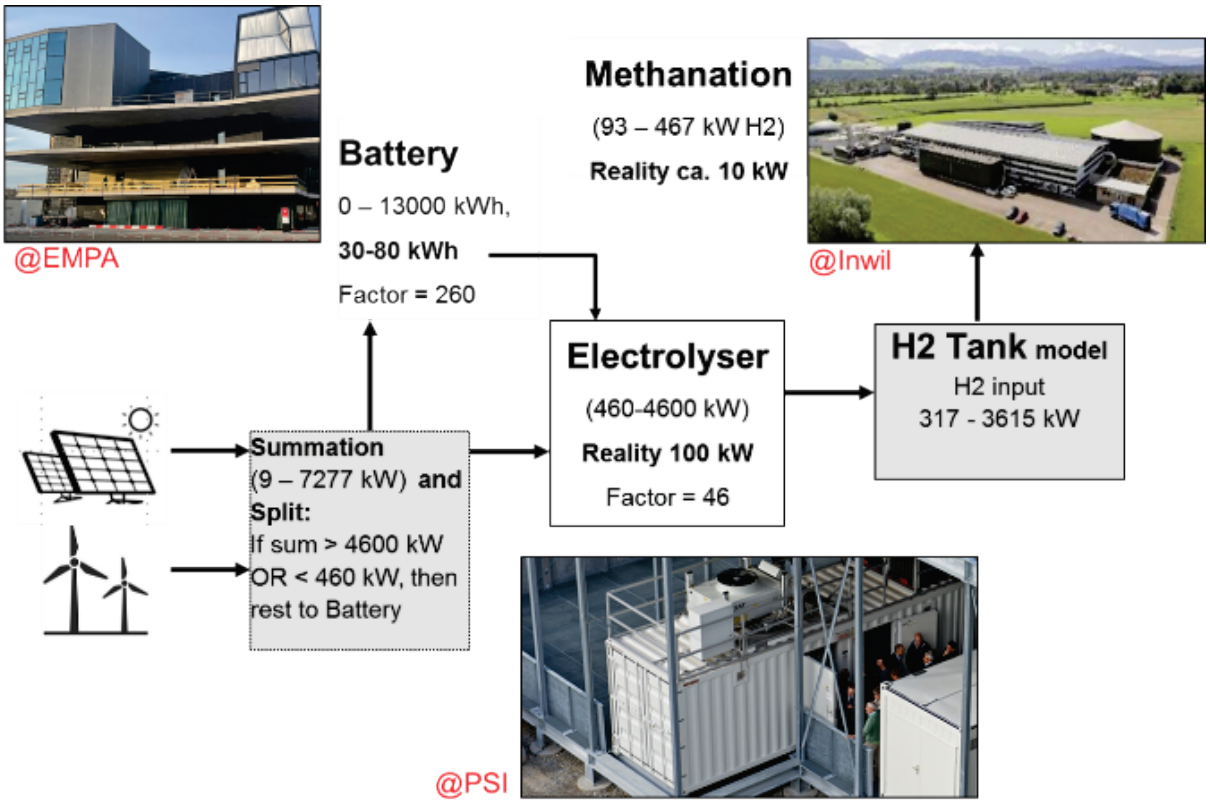


Figure 17: Topology for the experiment.

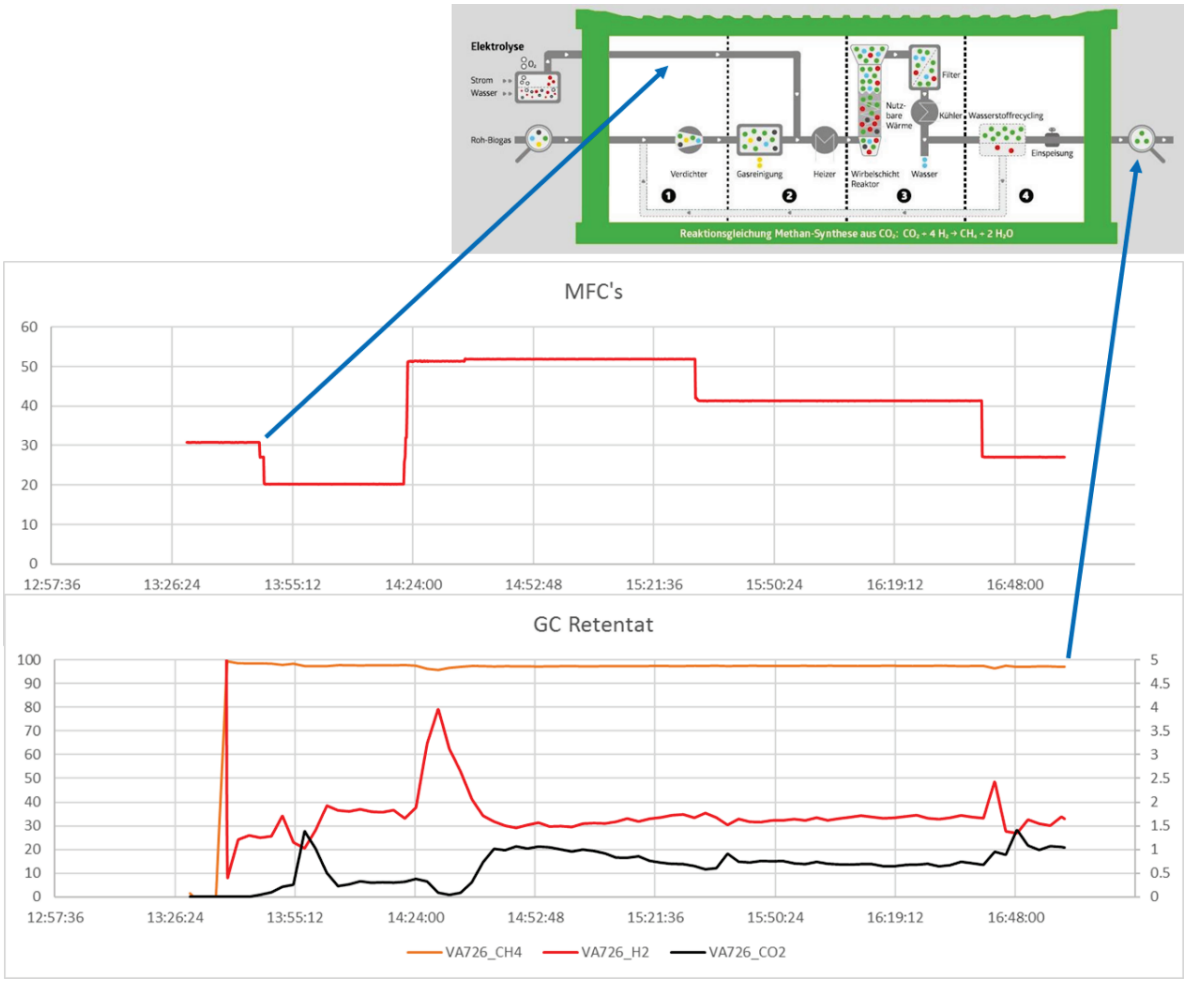


Figure 18: Part load study of PSI's TRL5 fully automated methanation plant (with integrated gas cleaning, upgrading membrane and recycle loop), connected to the biogas plant in Inwil. Variation of the hydrogen feed flow rate (upper diagram, between 45% and 100% of the nominal capacity) and resulting concentrations of methane (VA726_CH4, left y-axis), hydrogen (VA726_H2, right y-axis) and carbon dioxide (VA726_CO2, right y-axis) in the injection line to the gas grid

3.4 Electric vehicle fast charging station at a highway (Subproject T3.10)

Within Subproject T3.10 a similar problem was studied as in Subproject 3.9, namely a highway rest stop with a fast charging site for battery electric vehicles (BEVs) [39]. With increasing share of BEVs in the passenger car fleet, the fast charging demand may render the electric system overloaded. A possible solution is again peak shaving using some electricity storage. In this analysis, the focus lies on a PEM electrolyzer, a PEM fuel cell system and a hydrogen tank as storage media (see concept in Figure 19 and actual setup in Figure 20.

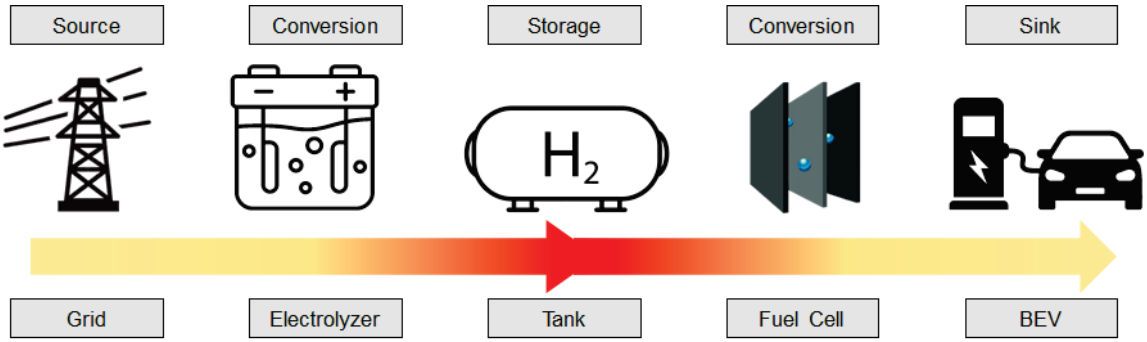


Figure 19: Schematic power flow from the electric grid towards a fast charging site at a highway rest stop, potentially buffered, using a storage link with a Hydrogen Energy Storage System (HESS).

The results, found in the case of the single charging events tests, prove the effectiveness of the HESS in peak shaving the BEV fast charging demand at a highway rest stop. The yellow curve in Figure 21 shows the fast charging demand (FCD) using the virtual battery of ReMaP. The values of that curve are scaled measurements of the EMPA battery's AC power.

The main discriminator is the maximum quarter-hourly average power. For the fast charging demand itself (yellow signal), this occurs in the time section after 14:15 h, where the maximum is around 100 kW (indicated by the yellow arrow). Comparing this value with the peak shaved power signal (in red), one finds the maximum average power of 60 kW over 15 minutes (indicated by the red arrow), which is comparably much lower. This was achieved, using a rule based on / off algorithm, which depends on the value of the peak shaving limit shown with the black line in Figure 21.

Since these maximum average values translate directly into the demand charge costs, above found results indicate the cost saving potential, which has however be balanced with the additional costs of the HESS system. In addition, this proves, that it is technically feasible to peak shave the fast charging demand using a HESS.

Figure 20: Physical system overview for HIL tests

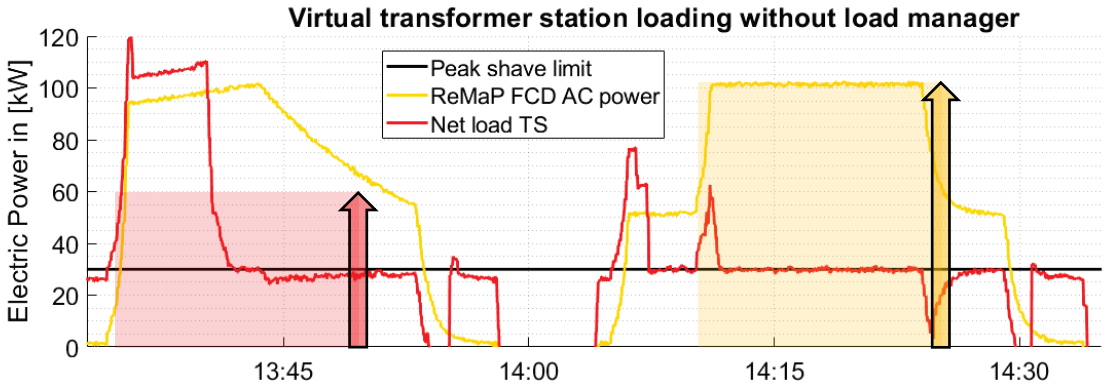


Figure 21: Single fast charging events, scaled to ReMaP battery capacity

4 Results of subprojects

The ReMaP project consisted of two main pillars, the development of the ReMaP platform (see previous Section 2) and a number of Subproject that tackled relevant questions in the context of future energy systems. Each of these subprojects is documented in a dedicated report, this section gives a quick overview of motivation and main results.

4.1 Distributed State Estimation and System Operation (Subproject T3.1)

The introduction of renewable energy sources and flexible demand, especially in distribution grids, opens new paths toward more sustainable power systems. Yet, the volatility and unpredictability of these elements, e.g., household demand or available renewable power, pose severe challenges on the reliability of the grid. Therefore, a faster and more efficient grid operation is essential for creating a more sustainable power system.

Online Feedback-based Optimization (OFO) approaches have shown very promising results to operate the grid in real time under time-varying conditions. Instead of operating the grid by solving an Optimal Power Flow (OPF) problem, an OFO controller uses measurements as feedback to continuously drive the set-points of the controllable elements towards the OPF time-varying solutions. OFO is able to enforce grid safety constraints, e.g., voltage limits; it requires minimal knowledge of the grid model, only the steady-state input-output sensitivity; and can operate in real time, e.g., at sub-second control-loop rates. Different variants of OFO controllers were developed in Subproject T3.1 and integrated and tested in ReSIM.

The first variant considered an OFO approach with noisy measurements, and without full state information, i.e., without direct measurements of some grid states of interest. To generate the state information required by the OFO controller, a connection was made to a dynamic state estimation that uses available measurements to produce the necessary estimates. Under simplifying model assumptions, in this case a linear power flow model approximation, the stability of the OFO-estimation interconnection could be verified. Also its convergence towards the optimal set-points and the boundedness of the resulting stochastic error could be shown. Finally, the performance of the approach could be demonstrated on a numerical simulation using the IEEE 123-bus test feeder.

The second variant considered again a case with noisy and incomplete measurements, and a combination of OFO with a dynamic estimation. However, in this case, a nonlinear power flow model was used within the estimation, instead of considering an approximate linear model and risking a performance degradation as consequence. Specifically, a Newton's method based online power flow solver was used, which allows to approximate the power flow solutions dynamically in a computationally efficient way that is suitable for real-time system operation. Additionally, a quadratic-programming-based OFO controller was employed that is able to handle grid constraints more efficiently, and limits the maximum possible violation. Again, the convergence and effectiveness of the OFO-estimation interconnection with an online power flow solver could be shown with a numerical simulation using the IEEE 123-bus test feeder. Here, a performance improvement over a projected-gradient-based OFO could be observed.

As last variant, a model-free OFO was studied that learns the input-output sensitivity. An inaccurate sensitivity, hence a model-mismatch, may lead to suboptimal decisions and jeopardize the performance of OFO. Therefore, a standard OFO controller was combined with a recursive least-squares estimation to learn the sensitivity from measurements during real-time operation. For that, (i) the sensitivity was modelled as a time-varying parameter; (ii) the change in the outputs caused by a change of the controllable inputs was used as measurement; and (iii) sequentially the sensitivity was updated using a Kalman-filter-based approach. A small white-noise signal in the OFO controller was introduced to guarantee persistency of excitation, which enables the controller to explore infinitely many directions, and allows to analytically certify convergence of the OFO with sensitivity estimation. Again, the performance of the approach was validated on a numerical simulation using the IEEE 123-bus test feeder, showing superior performance compare to a state-of-the-art OFO with a constant sensitivity approximation.

Details of the results can be found in the full project report [1].

4.2 Probabilistic load forecast (Subproject T3.2)

The transition to what is commonly referred to as the “smart grid” in essence comprises the deployment of additional sensing technology (including smart meters) and the actuation of various controllable devices in the low-voltage system. The additional sensing provides the opportunity to evaluate and access the flexibility of a consumer and to bring more transparency to the low-voltage grid on the one hand, but on the other hand also raises the issue of privacy concerns. Hence, there is the competing desire to improve the monitoring and predictability of the system in order to optimally use its flexibility while at the same time protecting the consumer from providing a level of information to the utility that allows extracting private information.

For better grid operation, the use of short-term predictions, be it minutes or days in advance, at the customer level is of great help. The granular data collected via smart meters and appliance sensors can be used as input to machine learning approaches that then provide predictions for the desired time range. At the transmission level or highly aggregated level, deterministic forecasting algorithms can properly predict the load which is relatively smooth and periodic. However, the load in distribution grids, and especially at the customer level, is highly volatile such that deterministic approaches are not satisfactory. Hence, due to the inherent uncertainties and inaccuracies of predictions, the approach should provide probabilistic predictions, i.e., distributions and probability quantiles instead of deterministic values. Probabilistic forecasting algorithms can indeed cover the entire uncertainty range without assuming specific error distributions and can update their prediction in real time.

In addition, amongst the measures to facilitate grid operation and thanks to the wide-scale rollout of smart meters and advance metering devices, Distribution System Operators can potentially make use of the flexibility of their customers. This presupposes that the consumption of certain flexible loads can be curtailed or shifted, either via direct load control or via price signals. This is known as Demand Side Management (DSM). For instance, domestic devices such as heat pumps, water heaters, wet appliances (i.e., dishwasher, washing machine, dryer) and cold appliances (i.e., refrigerator, freezer) can provide flexibility to the system operator without considerably impacting the comfort of the users.

Subproject 3.2 focused on three specific aspects: (i) Existing work of load forecasting was extended to a probabilistic method, i.e., instead of providing point forecasts, we provide quantile forecasts that capture the uncertainty in the prediction of the consumer load. (ii) We leveraged the fact that high-resolution load measurements likely also provide information about the flexibility of a consumer, and we designed data-based control algorithms for DSM of controllable devices with flexibility potential. (iii) We studied how the design of prediction and DSM control algorithms can co-exist with ensuring sufficient privacy of the consumer. For this work we leveraged the fact that real data was available from ReMaP experiments.

The main findings of Subproject T3.2 were:

- Probabilistic load forecasting algorithms combine online and offline learning. Historical data were used to train multiple regression models, generating independent forecasts. These models were then combined into an ensemble model for enhanced accuracy. Both deterministic and probabilistic models were applied to both building and individual appliance levels. Results showed that deterministic models are less effective due to load volatility, while probabilistic models provided improved uncertainty intervals. This approach was integrated as a software package into ReSIM to predict building-level power flows.
- Based on smart meter data, flexibility envelopes, representing temporal flexibility potential, were quantified using machine learning and probability distribution function-based approaches. The results indicate that combining forecasting and approximation models offers a promising solution to accelerate flexibility quantification without significant loss of precision.
- Finally, we explore various solutions to protect user privacy on data collected by smart meters while enabling effective data usage. Results suggest that the use of home battery with lower data granularity can enhance privacy protection without substantially compromising forecasting performance.

Details of the results can be found in the full project report [6].

4.3 Design and operation of multi-energy systems (Subproject T3.3)

With the proliferation of distributed energy resources (DER) such as photovoltaic generators, batteries and other smart grid devices, the integration and complexity of energy distribution systems increases, providing challenges and opportunities for reliable and resilient energy supply. The Subproject T3.3 assesses and optimizes future neighborhood energy systems, focusing on reliability and resilience, and assessing specifically Active electricity Distribution Networks (ADNs) and Multi-Energy Systems (MES).

Regarding ADNs, the following steps were taken: (i) analyzing the reliability of current electricity distribution networks, (ii) developing a design method for ADNs within the energy trilemma of cost, greenhouse gas (GHG) emissions and security in terms of reliability and resilience, and analyzing the energy trilemma trade-offs for ADNs worldwide; and (iii) assessing the impact of ADNs on their hosting networks.

The reliability and flexibility of neighborhood MESs was studied by: (i) developing a MES operational and reliability optimization model and calibrating the model using measurement data from ReMaP demonstrator technologies; (ii) analyzing neighborhood MES reliability; and (iii) developing a MES flexibility model to assess the ability of MES to provide ancillary services to the electricity grid and absorb uncertainty related to forecast errors of electricity demands and renewable generation.

Design methods for ADNs were developed by combining different techniques: Using ReSIM, a Global Sensitivity Analysis (GSA) followed by an Uncertainty Analysis (UA) was performed on the model's input parameters. Additionally, the geospatial uncertainty of the ADN was considered through the definition of global energy regions. They were the result of a clustering analysis of global real geographical data: photovoltaic potential, wind power density and climate zone to assess the load profile. The GSA was used to initially list all the input parameters and then to screen the most influential to the optimal ADN design. Subsequently, those selected were analyzed more in detail in the UA analysis, through the definition of uniform and PERT probability distributions. The result were 20'000 ADN optimal solutions that were analyzed to spot the favorable conditions for ADN deployment as well as to find trade-offs between cost, GWP, and reliability, what we call the energy trilemma.

The important question of the interaction of ADNs with their hosting network was studied again using ReSIM, focusing on the impact of ADNs and electrification on voltage deviations, branch and transformer loading in a rural, a semi-urban and an urban hosting network is investigated. While the additional load due to electrification of heating and mobility can lead to voltage constraint violations in rural networks and to short-term branch overloads in semi-urban and urban networks, transitioning 50% of the load buses to the previously designed ADNs can solve voltage and overload issues in all investigated networks with 2 complementary mechanisms. Namely, on the one hand, the distributed energy resources (DERs) within ADNs enable a higher self-consumption, which reduces the total load in the network. On the other hand, an energy management system was developed and tested in ReSIM, which uses the available DER capacity to i) mitigate the imbalance between load and generation caused by day-ahead forecast errors and ii) to alleviate voltage and overload issues in the hosting networks.

The potential of MES in (1) offering flexibility to the surrounding electricity grid in the form of frequency control reserves and (2) mitigating short-term demand and renewable generation forecast errors was also studied. The results showed that the potential of MES in providing flexibility is significant, with 65-75% of its potential capacity being offered for frequency control reserves for at least three quarters of the year. An optimal design of MES however required additional incentives for technology investments. In an optimally designed MES with a reliability incentive of 1 hour islanding, Battery Storage Systems (BSS) were found to play a key role in providing flexibility provisions as they offered on average 80% of their discharge capacity as reserves throughout the year and in addition absorbed 100% of the internal load and generation uncertainty. A sensitivity analysis considering changes in market frameworks and technology costs showed that reserve provision by itself does not incentivize additional technology investment. However, reserve provision with existing technologies is optimal for a wide variety of market frameworks and technology costs and thus, indicates robustness to future market price and technology costs development.

Details of the results can be found in the full project report [14].

4.4 Virtual Neighborhood Load Emulation (Subproject T3.4)

The transition towards decentralized energy supply will involve more technologies operating at the building and district levels. Therefore, carefully constructed future energy demand scenarios at the district level are required in order to effectively design, optimise and operate such energy supply systems. In this perspective, project VILLE presented the development and analysis of future energy demand projections in Swiss districts, taking into consideration urban development scenarios, climate change mitigation scenarios, and building envelope and systems retrofit scenarios.

Three Swiss district archetypes were formed based on the spatial division method of the Swiss Federal Statistical Office, where Swiss municipalities are categorized into three main groups: urban, sub-urban and rural. This work focused on three case studies (the districts of Altstetten, Echallens, Airolo) to represent each district, respectively. Three distinct urban development scenarios were then formed (Business-As-Usual (BAU), Digitalization (DGT) and Polycentric Urban Network (PUN)), taking into consideration nation-wide urban development trends such as the inward development and densification strategies, future population growth or decline, and the digitalization trend where flexible work conditions would decrease demand for offices and drive their conversion into mixed-use or residential spaces.

The climate change scenarios were chosen for the years 2040 and 2060, as the energy systems designed today (starting from 2020) are likely to be operating for up to 50 years. The projections selected represent a business-as-usual scenario, in terms of greenhouse gases emissions mitigation, and a rapid mitigation scenario that limits the warming of global mean temperature to 2°C.

The building envelope and building systems retrofit scenarios were developed based on the Swiss Energy Strategy 2050, which outlines the aim of the Swiss government to reduce the energy consumption of buildings by 45% by 2050. These retrofit scenarios comprised of a series of foresights on how the Swiss buildings could be performing in the future, with regards to their envelope and systems and also how built environment policies as well as the occupants will respond to the climate change via the installation of space cooling systems as well as the alteration of space cooling and heating setpoints, in an adaptive manner.

The simulations of the district archetypes were performed using the City Energy Analyst (CEA), an open-source computational framework for the analysis of urban energy systems, which uses dynamic and multi-physics models to forecast the load profile of heating, cooling, and electricity of single and multiple buildings up to entire districts, in hourly time-steps. In addition, a plug-in for building stock transformation was developed to generate input scenarios for energy demand simulation.

The results showed that each district presents unique challenges. Urban districts can develop a demand for space cooling comparable to space heating by 2060, while rural districts would remain heating dominated depending on the location. Sub-urban districts present the greatest uncertainty in terms of demand reduction, with urban development as well as retrofit decisions playing a pivotal role in the end result. Overall, urban development scenarios can play a very significant role in energy planning in districts, and if not taken into account, any future energy demand projections might be unrealistic and thus misleading, leading to great risks for the local energy grid. Climate change can have a significant effect on future demand for space cooling demand across all district archetypes, while the effect on space heating demand was found to be more prominent in rural districts. Building envelope retrofitting and building systems replacement can lead to a 45% reduction in energy demand by 2050, as well as significantly reduced thermal discomfort if the retrofit rates for building envelopes and those for systems are at least 3% and 10%, respectively.

The main findings of Subproject T3.4 were:

- Climate change was found to have a significant effect on future demand for space cooling demand across all district archetypes, both in terms of total annual demand increase but also in peak demand. Peak total demand coinciding with peak cooling demand in the summer months. At the same time, the effect of climate change on space heating demand was found to

be more prominent in rural districts, where the reduction of heating degree days in future years can be substantial.

- Building envelope retrofitting and building systems replacement can lead to significant reduction in the energy demand, allowing Switzerland to achieve its targets for 45% reduction by 2050 as set in the Energy Strategy 2050. However, for this to be achieved, the retrofit rates for building envelope need to be at least at an annual of 3%, while the replacement of building heating systems with modern efficient configurations (heat pumps) need to be nearing an annual rate of 10%. Moreover, with a 3% uptake of highly efficient space cooling systems, thermal discomfort can also be kept at minimum levels, given an adaptive behavior towards the warming climate.
- Urban Development scenarios play a role of significant importance that should not be overlooked when considering energy planning in districts and neighborhoods. The composition of the building stock, the growth in population as well as the density of occupation, should be an inherent part of any planning, since if not taken into account any future energy demand projections might be unrealistic and thus misleading, leading to great risk for the local energy grid.
- The districts archetypes analyzed in this work can potentially represent a large part of Switzerland. The results showed that by 2060, in the urban district archetype, space cooling demand can reach equivalent amounts to space heating demand, leading to important considerations with regards to systems design and optimization in these areas. In the sub-urban district archetype, the findings revealed that the energy planning can prove to be very challenging, depending on the urban development trajectories as well as the energy efficiency strategies. Contrary, in the rural district archetype, these difficulties were found to be less pronounced, with space heating demand possibly dominating the decision-making process, yet much depending on the local regional climate.

Details of the results can be found in the full project report [21].

4.5 Improved combined heat & power plant concepts (Subproject T3.5)

Combined heat and power (CHP) plants convert chemical energy, stored in either fossil or renewable fuels, into thermal and electrical energy. Such plants, depending on their size and converter type (internal-combustion engine (ICE), gas turbine, fuel cell, Stirling engine, etc.), can be started up quickly and have great potential to contribute to a stable future energy system based largely on fluctuating renewable energy sources. Conventional combustion-based CHP plants convert exergetically valuable high-temperature heat (at 500 – 700°C) in the exhaust gas to low-temperature heat (at 30 – 80°C). Therefore, a conventional ICE-based micro-CHP (mCHP) plant features a steady-state theoretical exergy efficiency of only 40-60% (depending on its size) although the energetic efficiency can be as high as 100% if condensation in the exhaust takes place. If the high-temperature thermal power was used directly at the available temperatures instead of converting it to low-temperature heat, the exergy efficiency could be improved.

The flexibility of a CHP plant rests on its ability to be started up and shut down quickly and is strongly linked to the heat-demand profile of a consumer and the capacity of the thermal energy storage (TES). The exergy efficiency of a CHP plant and buffer storage combination with a given heat-demand profile is not only dependent on the steady-state exergy efficiency of the CHP plant, but also on the storage capacity and its thermal losses. If further components are necessary to extract heat from the storage (e.g., a heat pump to extract heat from a borehole), the efficiencies of these components and the temperature levels of the fluids need to be considered, too.

As CHP plants could play an important role in future energy systems, the overall goal of Subproject T3.5 was to find a way to increase both exergy efficiency and flexibility of such plants. Additionally, the ability to store energy seasonally was investigated. The focus was on mCHP plants with an electrical output power of below 10 kW but the developed concepts should also be able to suit bigger plants to a certain extent.

Two options to improve the flexibility and exergy efficiency of a conventional combined heat and power (CHP) system were investigated with simulations. The more promising option was further tested with hardware-in-the-loop (HIL) experiments on the ReMaP platform (see also Section 3.1). It was seen that a steam methane reformer (SMR) CHP concept is to be favored against the concept of combining the CHP with a ground thermal storage system and a heat pump. Even though the latter has an infinite potential for flexibility, its efficiency suffers massively from thermal losses through diffusion in the ground. Simulations showed that over the course of a year, the SMR-CHP concept increases the degree of self-sufficiency in electricity demand by 30.7% and the exergy efficiency by 8.6% relative to the conventional CHP system.

A standard heat-led control concept for conventional CHP systems was improved by also considering economic aspects and using linearized models of the investigated systems to optimize control parameters. For the conventional and the SMR-CHP system, the operating costs were reduced by 1.9% and 2.1% while the degree of electrical self-sufficiency was increased by 3.8 and 1.7 percentage points at the cost of reducing the exergy efficiency by 0.23 and 0.5 percentage points respectively. Both hardware and software components were integrated into the ReMaP platform successfully. Subsequent HIL experiments with real demand data from the NEST building at Empa proved that the optimized control concepts for both the conventional and the SMR-CHP system are working in practice when applying a moving average filter to the electricity demand.

A prototype SMR-CHP plant was sized, designed and built on the basis of the Aladin II mCHP plant. First experiments showed that the chosen SMR components are not able to withstand the pulsations of the exhaust gases in combination with the very high temperatures of up to 750°C. Three constructive and operational improvements of the system were identified during the experiments.

The main findings were:

- The combination of CHPs with ground thermal storage is not recommended due to high diffusion rates.

- Over the course of a one-year simulation, a steam methane reformer CHP system promises to be more flexible (increased degree of self-sufficiency by 14.0 percentage points) and exergy efficient (increase of 2.9 percentage points) than a conventional system.
- The chemical and thermal performance of an SMR-CHP system still needs to be validated in a follow-up project, because the chosen components are not able to withstand the rough conditions of a single-cylinder IC engine.
- A hybrid control approach that expands the thermostat control to favour CHP dispatching when the economical break-even point is reached can improve the degree of self-sufficiency and reduce operating costs (by 3.8 percentage points and -1.9% respectively for the conventional CHP).

Details of the results can be found in the full project report [25].

4.6 Battery storage (Subproject T3.6)

Batteries will be a key component of future energy systems – in battery electric vehicles and as stationary electricity storage devices. The goal of Subproject 3.6 was to gain insights into the factors that contribute to battery degradation and to improve the longevity of batteries targeting stationary applications. To achieve this, a cycling aging campaign was conducted to simulate real-world conditions, including both standard and "abuse" scenarios. Four different temperatures were evaluated (25°C, 50°C, 0°C, and -10°C) and two different charge/discharge rates (1C/1C and 3.3A/10A) were studied, along with three different Depth of Discharge (DOD) levels (25-75% SOC, 0-50% SOC, and 50-100% SOC). The study monitored various parameters during the aging process and analyzed the efficiency loss per cycle and the discharge capacity loss to predict the life of the battery cell.

The results revealed a strong influence of temperature on the performance of all three technologies (NMC/LTO, NMC/Gr, and LFP/Gr). The correlation between cell voltage, percentage of delivered capacity, and cell skin temperature evolution was clearly visible, indicating that cell performance is highly sensitive to temperature changes. Sudden changes in cell voltage and cell skin temperature were strongly interdependent. Low temperature conditions can limit ionic conductivity and reduce the overall performance of the cell.

For standard C-rate cycling, NMC/LTO was found to be the superior choice for applications that require a standard C-rate, stable performance, longer battery life, and higher delivered capacity. For high C-rate cycling, both technology and temperature play a critical role in determining long-term performance and stability. At low temperatures, LFP/Gr performed better than NMC technologies, but at elevated temperatures, NMC/LTO was the better choice for stability, while NMC/Gr performed better at intermediate temperatures of 25°C. Calendar aging and cycling at elevated temperature (50°C) resulted in faster degradation for LFP/Gr compared to NMC cells, making NMC a better choice for high temperature applications.

It is crucial to consider the long-term stability and degradation behavior of these technologies, as well as any potential safety issues, such as thermal stability, when making a final choice for a specific application. Careful consideration of the operating conditions and material selection is essential to ensure optimal performance and lifetime of the battery.

The main findings of Subproject T3.6 were:

- Temperature had a strong influence on the performance of all three cell technologies (NMC/LTO, NMC/Gr, and LFP/Gr) studied. Depth of Discharge (DOD) had a noticeable impact on the degradation process of all three technologies, especially at 0°C and 25°C cycling aging.
- NMC/LTO technology is a superior choice for applications that require a standard C-rate, stable performance, longer battery life, and higher delivered capacity.
- For high C-rate cycling, the choice of technology and ambient temperature play a critical role in determining the long-term performance and stability of the battery. NMC/LTO was found to be the better choice for stability at elevated temperatures, while NMC/Gr performed better at intermediate temperatures of 25°C.
- Calendar aging and cycling at elevated temperature (50°C) resulted in faster degradation for LFP/Gr compared to NMC cells, making NMC a better choice for high temperature applications.

Details of the results can be found in the full project report [26].

4.7 Capacitive energy storage (Subproject T3.7)

Energy storage devices that can provide fast energy uptake and delivery, constitute an essential research and technological direction towards the realization of a sustainable energy future, by providing intermittent storage of energy from renewable sources, such as sun and wind. State-of-art commercial supercapacitors rely on the physical mechanism of charge storage, called electrical double layer capacitance (EDLCs). Therefore, they offer excellent high power capabilities characterized by at least an order of magnitude higher power densities than that of batteries, coupled with the exceptional cycle life of close to 100,000 of cycles when exploited according to the manufacturer specifications temperature, voltage and current ranges.

Owing to these characteristics, use of supercapacitors as a part of energy frameworks can provide exceptional benefits related to (i) overall increase in energy efficiency of the processes through recovery of the otherwise wasted energy (i.e. in regeneration braking systems) and (ii) help mitigate limitations of the current battery technologies related to the slow charging and limited cycle life through tandem integration. Yet, while advantages of supercapacitors are generally clear, several knowledge/information gaps remain and need to be addressed in order to formulate clear recommendations related to their use as a part of grids or micro-grids.

To achieve this it is crucial to evaluate the performance of commercially available devices under relevant operation conditions. Therefore, Subproject T3.7 aimed at: (i) conducting a comprehensive electrochemical behavior assessment of commercial supercapacitors, (ii) identifying critical parameters for maintaining stable cycling performance, (iii) and evaluating the potential integration of supercapacitors into battery-based grid storage systems.

The experimental work included: (i) Aging tests at a different usage scenarios (temperature, voltage, currents) of supercapacitors to obtain evolution of capacitance, internal resistance as well as leakage currents, (ii) Heat generation evaluation under different usage scenarios to formulate recommendations of the thermal management during operation; and (iii) Investigation of supercapacitor application as “power buffers” to batteries, evaluation of battery lifetime extension depending on the usage scenario (including extreme conditions), and recommendation on supercapacitor usage for transient energy storage in tandem with batteries.

Subproject 3.7 revealed critical insights into the lifecycle of commercial supercapacitor systems, underscoring the variability in performance across products and operational scenarios. The research indicated that even with similar specifications, supercapacitor cells from different manufacturers can degrade at markedly different rates, stressing the need for independent testing prior to their use in grid storage. A notable finding is the approximately 5% cell-to-cell capacity variation, which can cause uneven current and voltage distribution when cells are assembled into modules, potentially leading to operation beyond manufacturer-recommended conditions. Furthermore, cells subjected to just slightly above recommended voltage degrade at a significantly faster rate. The study also highlighted the detrimental effect of high currents on capacity retention, a situation that worsens with increased temperature. Thermal imaging has shown that high current operation leads to substantial rises in cell temperature, pointing to the necessity of efficient thermal management in module design. Lastly, the integration of battery-supercapacitor modules has demonstrated potential in enhancing battery life by minimizing stress through narrower state-of-charge ranges, offering a promising direction for extending the longevity of energy storage systems.

The main findings of Subproject T3.7 were:

- Differential Degradation Rates: Supercapacitor cells from different manufacturers with identical specifications can have vastly different degradation rates, necessitating independent testing for grid storage selection.
- Capacity Variation Among Cells: Individual supercapacitor cells exhibit ~5% variation in actual capacity compared to specifications, leading to non-uniform current/voltage distribution in modules.

- Impact of Overvoltage: Continual operation of supercapacitors at voltages slightly above the recommended level (3V) leads to a notable increase in cell degradation rates.
- High Currents and Capacity Fading: High operating currents accelerate supercapacitor capacity degradation, which is exacerbated at elevated operation temperatures.
- Thermal Management Needs: High current conditions significantly increase supercapacitor cell temperatures, necessitating robust thermal management systems for module integrity.
- Battery Life Extension: Pairing supercapacitors with batteries allows for a narrower state-of-charge range in battery usage, reducing stress and potentially extending battery cell lifespan.

Details of the results can be found in the full project report [27].

4.8 Advanced control algorithms for future energy systems (Subproject T3.9)

The shift in the energy landscape towards the penetration of fluctuating renewable energy sources and distributed energy storage devices opens technical challenges related to the need for ensuring the reliability of the energy system as a whole. Subproject T3.9 aimed at developing and analyzing advanced control algorithms for the optimal operations of future energy systems, with a specific focus on the integration between the power and mobility sector.

The work was divided into three main work streams: (i) Modeling and system identification of the ESI platform at PSI and the move platform at Empa; (ii) Developing advanced control algorithms for the optimal operations of the aforementioned systems with a particular focus on robustness, computational complexity and scalability of the proposed methods; (iii) Implementation of these algorithms in ReSIM and experimental validation on the platforms with HIL experiments.

On work stream (i) a modeling library with detailed models of the devices at the ESI platform and at the move platform was developed that can capture the nonlinearities of real-world hardware components and ultimately support the development of accurate model-based simulations and control schemes. These devices included electrolyzers, fuel cells, gas cleaners, compressors and storage tanks. On the demand side, we modeled the hydrogen usage of a fuel cell vehicle fleet as stochastic variable accounting for the probability distributions on the arrival time, the amount of hydrogen recharged and number of incoming daily vehicles. The distributions were fitted based on data from the move demonstrator at Empa.

Work stream (ii) provided advanced control algorithms for the optimal control of future energy hubs that are capable to deal with the inherent stochasticity affecting such systems and come with certificates in terms of robustness, computational footprint and scalability. An Online Optimization Algorithm was developed that solves an optimization problem for the real-time control of complex systems. The main focus here was on the computational aspect: we provide a dual dynamic programming framework for the control of nonlinear problems over long horizon (for example, for seasonal energy storage) with an optimal trade-off between accuracy and computation.

Another example is an Online Peak Shaving Optimization controller that allows to shave the peak of an EV charging station equipped with an hydrogen energy storage system. The aim here was twofold, namely to provide and test a controller that is capable to account for the inherent stochasticity of the EV charging process, and to understand whether energy-based technologies can be successfully employed for ancillary services.

A third example dealt with the demand side management of buildings, where we introduced a novel weighting scheme to account for cases where the controller optimization horizon spans multiple demand charge periods. The simulation study allowed us to compare various design and modeling alternatives, ultimately proposing a policy based on causal affine decision rules. The findings are published in [5].

Finally, the Online Peak Shaving Optimization controller was tested using the ReMaP platform in a HIL experiment on the ESI platform of PSI. These tests confirmed that the performance was superior to existing control schemes (see also Section 3.2).

The main achievements and findings of Subproject T3.9 were:

- The development of detailed models for both platforms that capture the complexity of real-world equipment, hence enriching the model library of ReMaP;
- The development of advanced controllers to optimally operate both platforms in order to predefined performance metrics (for example, peak shaving or profitability/sustainability tradeoff).
- The validation of the controllers in the platform via the ReMaP simulation framework, hence offering a path from software to hardware.
- To demonstrate the need for and the effectiveness of advanced control algorithms to tackle the technical challenges brought in by the energy transition in terms of uncertainty handling of both

source and supply side (robustness) and ability to handle large-scale problems (for example, robustness long-horizon control problems);

- To demonstrate the effectiveness of hydrogen-based storage to integrate fluctuating renewable energy sources and provide ancillary services to the main grid.

Details of the results can be found in the full project report [30].

4.9 Electrochemical energy storage (Subproject T3.10)

Currently, large scale electricity storage can mainly be accomplished by pumped hydro power storage plants that allow storing electricity from peak times of production to times when the demand exceeds the actual production. While in the past, this technology was used to store continuously produced nuclear energy from night to the demand peaks over day, in the future (with high shares of renewables, especially photovoltaics) this technology can help to shift electricity from production peaks around noon to the evening hours. A similar service can be achieved with batteries, such as the one installed by EKZ in Volketswil (18 MW, 7.5 MWh). Both technologies, however will economically not be able to store electricity for longer times, especially not from summer to winter which will be a major challenge in future.

Power-to-Gas is an option to convert electricity otherwise not needed by means of water electrolysis to hydrogen, a versatile and clean chemical energy carrier. This technology is investigated in different demonstrator projects such as ESI (PSI), move (Empa) and the Hybridkraftwerk Aarmatt in Zuchwil. Meanwhile, first companies offer fuel cell driven trucks and the according system of fueling stations. Large peaks in electricity production are expected for utilization of photovoltaics which ask for storage capacity with large power. Covering the peak only by electrolysis would lead to large installed electrolyzer power (and the inherent, high capital costs) with relatively short operation times. Batteries can help to decrease electrolyzer capacity and to prolongate times of higher electrolyzer capacity usage. On the other hand, use of the produced hydrogen in other sectors and conversion into energy carriers with already existing distribution and storage infrastructure may facilitate the seasonal storage.

Within Subproject T3.10, three different case studies were conducted to better understand how the cascade of energy storage options (battery, electrolysis/H2-tank, methanation/gas grid) can be used in a synergistic way to handle the challenges of the intra-day and seasonal imbalance of electricity demand and supply by volatile renewable sources such as PV, wind and hydropower.

For the grid-neutral and renewable hydrogen production for a fuel cell vehicle fleet, an electrolyzer was considered together with a Power-to-Gas seasonal storage system, which consists of a methanation, the gas grid as intermediate storage and a steam reformer. The dynamic operation of the plant over the year was simulated. The simulation study showed that in the next decades further decreasing costs for the electrolysis and the PV system as well as increased price for hydrogen make the grid-neutral renewable hydrogen production economically feasible.

For the experimental campaign, the electrolysis, gas cleaning, and H2-tank at PSI, the container based methanation unit connected to a real biogas plant in Inwil as well as the battery at Empa, all connected via the ReMaP framework, were used for the experiments. It could be shown that the methanation plant is able to handle part load changes without compromising the gas quality (see also Section 3.3).

Concerning the local (peak) electricity generation hub for an electric vehicle fast charging station at a highway, it could be shown by simulations that already with today's battery prices, battery based electricity storage systems (BESS) are financially attractive to peak shave the battery vehicle (BEV) fast charging demand of the considered fast charging site. Even with today prices for ESS, the HESS solution becomes more cost efficient than a BESS with increasing demand charges and future demand.

During the experimental tests using the electrolyzer, the hydrogen tank and the fuel cell stacks at the ESI platform at PSI and the battery at Empa, it was shown that the HESS becomes a viable and validated solution for peak shaving applications (see also Section 3.4).

The system study considering the integration of local prosumage systems within the Swiss energy system allowed to quantify key conditions for profitable operations of such a P2X energy hub, which are the access to real-time wholesale market electricity price signals, access to a future retail hydrogen market, and an overall strong climate policy. The results also show that the Power-to-X operation leads to an increased deployment of local renewables.

Details of the results can be found in the full project report [39].

4.10 Connections between Subprojects T3.X and the ReMaP Platform

The ReMaP project had two pillars, (i) the development of the ReMaP platform with its two main constituents, the simulation framework ReSIM and the control framework that interfaces with real world experiments, and (ii) the Subprojects T3.1-10 that each dealt with specific aspects of future energy systems. This section gives a rough overview how the Subproject interfaced with the ReMaP Platform.

Subproject	**Topic**	**Connection to ReMaP platform**
T3.1	Distributed State Estimation and System Operation	Use of ReSIM for testing new control strategies. Transfer of software elements to ReSIM.
T3.2	Probabilistic Load Prediction for Optimized Grid Operation Capacitive Energy Storage	Use of ReSIM for testing new probabilistic load forecast strategies. Transfer of software elements to ReSIM.
T3.3	Design and operation of reliable, decentralized multi-energy systems	Use of ReSIM to simulate and control active distribution networks (including Adaptricity network simulation software).
T3.4	Virtual Neighborhood Load Emulation	Coupling of City Energy Analyst software with ReSIM. Use of ReSIM to simulate future cities (including Adaptricity network simulation software).
T3.5	Improved combined heat & power concepts	Use of full platform for HIL experiments on CHP plant
T3.6	Battery storage	Regular exchange
T3.7	Capacitive energy storage	Regular exchange
T3.9	Advanced control algorithms for future energy systems	Use of ReSIM for testing new control strategies, transfer of software elements to ReSIM, use of full platform for HIL experiments on electric vehicle charging
T3.10	Electrochemical energy storage	Use of full platform for HIL experiments on electric vehicle charging and hydrogen supply

5 Outlook

Besides generating scientific knowledge, a complex project like ReMaP with a strong experimental component leads to benefits for the participating institutions that last longer than the project. This section gives an overview.

5.1 Benefits for Venios energy platform

The progress made in terms of scaling and performance results from the successful application at ReMaP and provides valuable insights. The knowledge gained not only enables targeted improvement of current applications, but also helps to design innovative solutions and continuously increase the performance of the platform.

5.2 Benefits for ESI platform at PSI

ReMaP enables networked operation of the pioneering systems at PSI with systems at Empa and other locations via the cloud, so that external users as well as employees can control the systems outside PSI for their experiments.

Compatible programmable logic controllers in the individual systems ensure, based on HAZOP analyses, that the systems can only be operated within safe limits. The separate machine network, which is connected to the cloud via a head station with a master computer, enables secure, fast, precise and extensive communication. For professional networking via the Internet to the ReMaP Cloud with its extended functionality, the important elements of cyber security are integrated in the head station and the master computer according to the current state of the art.

The local, overarching control system at the head station at the PSI brings together systems that will be required in the future for the integration of renewable energy in the energy system. The local control system has a central control with a programming environment, visualization and data storage. More systems were included than planned in the proposal. Today electrolysis, fuel cells, methanation, micro gas turbine and hydrothermal gasification are included. But all data from PSI's building management system, such as PV, charging stations or building consumption, are also integrated, and can be used in real time.

The option for digital scaling is provided so that the systems, some of which are of different sizes, can be operated in a networked manner. Organizational and legal matters are ensured through a developed user agreement and a clearly defined ReMaP-wide process. As a result, in addition to the networked systems, the interdisciplinary knowledge required to build safety-relevant systems in networked remote operation is now available at PSI.

5.3 Benefits for ehub platform at Empa

Ehub's interfaces to connect to the ReMaP control framework have been developed during the ReMaP project and have been extended since. These interfaces are in daily operation and were used, for example, to interconnect ehub with the EPFL Smart Grid Demonstrator during the SFOE SWEET PATHFNDR project.

Beyond that, during the course of the ReMaP project, several NEST units have been planned, built and commissioned. Their building technology equipment (i.e., sensors but also actors and control units) have been integrated to ehub and made available specifically under the ReMaP project. The access to this static data (e.g., schematics, installation plans, equipment specification, etc.) as well as measurement data in one minute resolution will be continued beyond the end of the project together with the pre exiting infrastructure and accessed systems put in operation parallel to the ReMaP project. The described data is made available to interested students/researchers/developers under a process developed during the projects. This includes the possibility to not just read data but also to actively control installed systems via the provided interfaces for which a process has been defined as well. To increase the interpretability

of the provided data, a visualization environment using Grafana was configured and extended on a use case basis.

The PMU installations (phasor measurement units from Smart Grid Solutions) that were installed in the medium voltage ring of Empa's Dübendorf Campus will be adapted and extended under an Empa wide Campus Digital Twin project to facilitate net Zero energy solutions.

5.4 Further use of ReSIM

Within the ReMaP project, ReSIM was used (while being developed) to execute a set of “experiments”, each consisting of emulating the behavior of a selection of components which, in general, are part of the overall local-level energy system. Some experiments included hardware components (HIL simulation), while in others all components and control algorithms were emulated as ReSIM modules.

The significant experience gained by successfully implementing these experiments is currently leveraged by utilizing ReSIM as the simulation platform in other projects. The significant advantage of resorting to ReSIM is that it allows to very easily integrate the work of different researchers / engineers (typically consisting in the development of models and operational algorithms) into a software that allows the different pieces to seamlessly work together. ReSIM acts like the “motherboard” to which each module needs to be interfaced, thus ensuring the interoperability across the models.

ReSIM is used in the ongoing Era-net project “AI-assisted decision support for operational planning in distribution systems” (AISOP) as the platform where the following modules are connected:

- A solver of the AC power flow equations of electricity distribution networks.
- A module representing the energy management system of a building, optimizing the local resources (demand, generation and storage) to minimize energy cost.
- A module forecasting the future states (power injections / withdrawals and power flows) of an electricity network.
- A module setting dynamic tariffs to which the end-customers are exposed.

The above modules are operated in a closed-loop.

ReSIM is also used in the ongoing SWEET project “Pathways to an efficient future energy system through flexibility and sector coupling” (PATHFNDR), as a means to integrate the work performed by a large number of researches in a common software platform, hence allowing to emulate how the proposed system configurations and solutions would behave if they were to be implemented. A non-exhaustive of the types of modules which are to be integrated in ReSIM is provided below.

- Models of technologies: E.g.: heat pump, battery, electrolyzer, fuel cell, theral energy storage, thermal energy storage, hydrogen storage tank, heat exchanger, cooling system.
- Models of end demand: E.g.: thermal models of different types of buildings, demand for EV charging by different driver and at different types of charging stations (at home, at work, public).
- Operational procedures and/or control algorithms: E.g.: sector-coupled coordinated control of energy devices, charging schedule of fleet of EVs, clearing of local flexibility markets, bidding to flexibility markets, dynamic tariffs, closed-loop identification of required flexibility activation signals.

The above modules are to be used in combinations to each other in use-cases and case studies.

5.5 Further use of full ReMaP Platform

Currently, no plans exist to use the full functionality of the ReMaP platform to perform HIL experiments. However, the whole software suite has been properly documented (see Annex, Sections 7.1-7.4) and archived and can be reactivated when needed.

6 Publications

The following scientific publications were published within the ReMaP project. The list includes also the final reports of Subprojects T3.1-10. Note that Subproject T3.8 was related to the development of ReSIM which is described in Section 2.3 of the present summary report.

T3.1	[1] Final report Subproject T3.1 [2] M. Picallo, S. Bolognani and F. Dörfler, “Closing the loop: Dynamic state estimation and feedback optimization of distribution grids,” Electric Power Systems Research, September 2020. doi: 10.1016/j.epsr.2020.106753 [3] M. Picallo, D. Liao-McPherson, S. Bolognani and F. Dörfler, "Cross-layer Design for Real-Time Grid Operation: Estimation, Optimization and Power Flow", accepted for 2022 Power Systems Computation Conference, June 2022. [4] M. Picallo, L. Ortmann, S. Bolognani and F. Dörfler, "Adaptive Real-Time Grid Operation via Online Feedback Optimization with Sensitivity Estimation", accepted for 2022 Power Systems Computation Conference, June 2022. [5] PhD Thesis Miguel Picallo: “Interconnected Online Feedback Optimization and Estimation Algorithms for Power System Operation in Real Time”, March 2022.
T3.2	[6] Final report Subproject T3.2 [7] L. Von Krannichfeldt, Y. Wang, T. Zufferey, G. Hug, “Online Ensemble Learning for Energy Forecasting,” Master thesis, Power Systems Laboratory, ETH Zurich, 2020 [8] L. Von Krannichfeldt, Y. Wang, T. Zufferey, G. Hug, “Online Ensemble Approach for Probabilistic Wind Power Forecasting,” IEEE Transactions on Sustainable Energy, Vol. 13, No. 2, pp. 1221 – 1233, April 2022 [9] Y. Wang, L. Von Krannichfeldt, G. Hug, “Probabilistic Aggregated Load Forecasting with Fine-grained Smart Meter Data,” IEEE PowerTech Conference, Madrid, Spain, 2021 [10] T. Zufferey, G. Valverde, G. Hug, “Unsupervised Disaggregation of Water Heater Load from Smart Meter Data Processing,” MedPower, Cyprus, 2020 [11] T. Zufferey, G. Hug, G. Valverde, “Disaggregation of Cold Appliance Loads from Smart Meter Data Processing,” IEEE T&D PES Conference and Exposition Latin America, Montevideo, Uruguay, 2020 [12] S. Mohan, T. Zufferey, G. Hug, “Probabilistic Forecasting in Buildings and Privacy Concerns,” Master thesis, Power Systems Laboratory, ETH Zurich, 2020 [13] M. Jia, Y. Wang, Ch. Shen, G. Hug, “Privacy Preserving Distributed Clustering for Electrical Load Profiling,” IEEE Transactions on Smart Grid, Vol. 12, No. 2, pp. 1429 – 1444, March 2021
T3.3	[14] Final report Subproject T3.3 [15] R. Wu and G. Sansavini, ‘Balancing costs, emissions and security in Active Distribution Networks’, presented at the Applied Energy Symposium: MIT A+B (virtual), 2020, Accessed: Dec. 8, 2020. [Online]. Available at: https://www.research-collection.ethz.ch/handle/20.500.11850/450616 [16] Wu, Raphael, und Giovanni Sansavini. „Parameter Estimation for Distribution Grid Reliability Assessment“. In 2020 International Conference on Probabilistic Methods

	Applied to Power Systems (PMAPS), 1–6, 2020. https://doi.org/10.1109/PMAPS47429.2020.9183408. [17] Wu, Raphael, und Giovanni Sansavini. „Integrating Reliability and Resilience to Support the Transition from Passive Distribution Grids to Islanding Microgrids“. Applied Energy 272 (15. August 2020): 115254. https://doi.org/10.1016/j.apenergy.2020.115254. [18] Wu, Raphael, und Giovanni Sansavini. „Active Distribution Networks or Microgrids? Optimal Design of Resilient and Flexible Distribution Grids with Energy Service Provision“. Sustainable Energy, Grids and Networks 26 (1. Juni 2021): 100461. https://doi.org/10.1016/j.segan.2021.100461. [19] Wu, Raphael, Emanuela Bussino, Stefano Bracco, Silvia Siri, Paolo Gabrielli, und Giovanni Sansavini. „Optimization-Based Reliability Assessment of Multi-Energy Systems“. In Proceedings of the 31th European Safety and Reliability Conference, 7. Angers (France), 2021. [20] Wu, Raphael, und Giovanni Sansavini. „Energy Trilemma in Active Distribution Network Design: Balancing Affordability, Sustainability and Security in Optimization-Based Decision-Making“. Applied Energy 304 (15. Dezember 2021): 117891. https://doi.org/10.1016/j.apenergy.2021.117891.
T3.4	[21] Final report Subproject T3.4 [22] Popova, A., Hsieh, S., & Schlueter, A. (2022). Methodology for Scenario based Analysis of Future Energy Performance of Swiss Settlements at Urban, Sub urban, and Rural Scale. Submitted. Passive and Low Energy Architecture, Santiago. [23] Oraiopoulos, A., Hsieh, S., & Schlueter, A. (n.d.). Energy futures in representative Swiss districts under urban development, building retrofit and climate change scenarios. In preparation. [24] Hsieh, S., Thomas, D., Popova, A., & Oraiopoulos, A. (2021). A CEA plugin for the ReMaP / VILLE project [Python]. https://github.com/architecture-building-systems/cea-remap-ville-plugin
T3.5	[25] Final report Subproject T3.5 The completion of the project coincided with the retirement of Prof. Boulochos, therefore no papers were published
T3.6	[26] Final report Subproject T3.6 Three papers are in preparation.
T3.7	[27] Final report Subproject T3.7 [28] Chulgi Nathan Hong, Audrey B. Crom, Jeremy I. Feldblyum, Maria R. Lukatskaya, Metal-organic frameworks for fast electrochemical energy storage: Mechanisms and opportunities, Chem, Volume 9, Issue 4, 2023, Pages 798-822, ISSN 2451-9294, https://doi.org/10.1016/j.chempr.2023.02.016.Cqcwec [29] Dario Gomez Vazquez, Travis P. Pollard, Julian Mars, Ji Mun Yoo, Hans-Georg Steinrück, Sharon E. Bone, Olga V. Safonova, Michael F. Toney, Oleg Borodin and Maria R. Lukatskaya, Creating water-in-salt-like environment using coordinating anions in non-concentrated aqueous electrolytes for efficient Zn batteries, Energy Environ. Sci., 2023, 16, 1982-1991, DOI: 10.1039/D3EE00205E
T3.9	[30] Final report Subproject T3.9

	[31] B. Flamm, "Economic Model Predictive Control of Energy Storage", Ph.D. Thesis, ETH Zurich, 2020. Available at: https://www.research-collection.ethz.ch/handle/20.500.11850/444042 [32] Benjamin Flamm, Annika Eichler, Joseph Warrington and John Lygeros. «Dual Dynamic Programming for Nonlinear Control Problems over Long Horizons.» European Control Conference. 2018. [33] Benjamin Flamm, Annika Eichler, Roy S. Smith and John Lygeros. «Price arbitrage using variable-efficiency energy storage.» (Journal of Physics: Conference Series, CISBAT 2019) 1343 (2019). [34] Benjamin Flamm, Christian Peter, Felix N. Buechi and John Lygeros. «Electrolyzer modeling and real-time control for optimized production of hydrogen gas.» (Applied Energy) 281 (2021). [35] Benjamin Flamm, Guillermo Ramos, Annika Eichler and John Lygeros. «Multiperiod Stochastic Peak Shaving using Storage.» Submitted to Arxiv, 2020. [36] Marta Fochesato, Christian Peter, Lisa Morandi and John Lygeros. «Stochastic Peak Shaving for EV Charging Stations with Renewable Integration and Hydrogen-based Energy Storage.» In preparation, 2023. [37] Marta Fochesato, Philipp Heer and John Lygeros. «Multi-objective optimization of a power-to-hydrogen system for mobility via two-stage stochastic programming.» Journal of Physics: Conference Series, CISBAT 2021 2042 (2021). [38] Marta Fochesato, Timo Laaksonlaita, Philipp Heer and John Lygeros. «Modeling and real-time control of a hydrogen refueling station with uncertain demand.» Submitted to IFAC World Conference 2023. 2023.
T3.10	[39] Final report Subproject T3.10 [40] J. Witte, H. Madi, U. Elber, P. Jansohn, Tilman J. Schildhauer, Grid-Neutral Hydrogen Mobility: Dynamic Modelling and Techno-Economic Assessment of a Renewable-Powered Hydrogen Plant, manuscript in preparation [41] A. Gantenbein and T.J. Schildhauer, Challenges in part load operation of biogas-based power-to-gas processes (Part I), accepted for publication in Frontiers in Energy Research, section Bioenergy and Biofuels [42] Y. Wan, T. Kober, T. Schildhauer, T.J. Schmidt, R. McKenna, Martin Densing (2023): Conditions for profitable operation of P2X energy hubs to meet local demand under energy market access. Advances in Applied Energy, 100127, https://doi.org/10.1016/j.adapen.2023.100127

7 Annex

7.1 Specification of Control Framework

See File: SCS-Wiki-ReMaP - Specification-v2-20230224_151852.pdf

Specification

SGS REMAP

Exported on 02/24/2023

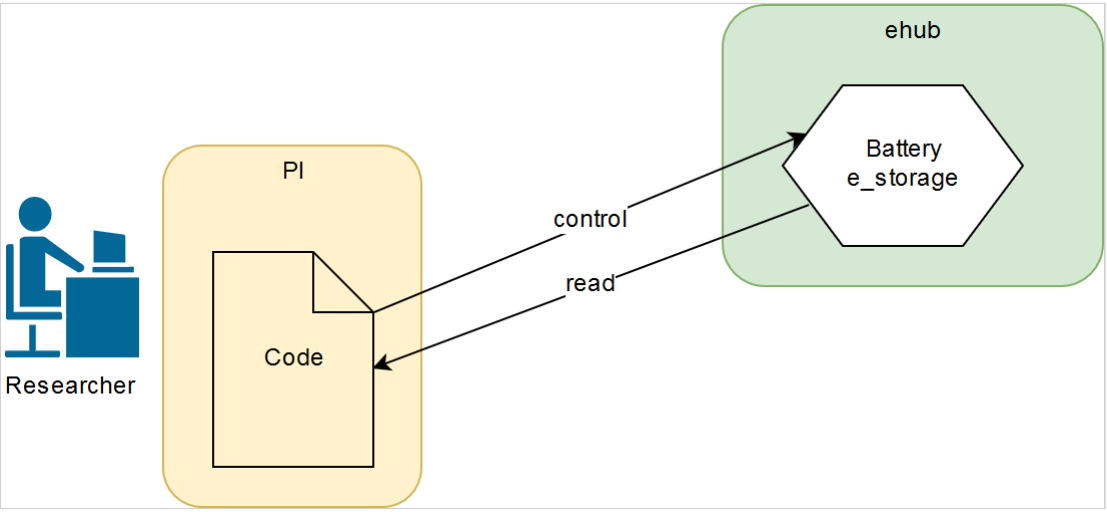
Table of Contents

1 Functional Requirements

1.1 Control of Hardware

1.1.1 Control at ehub

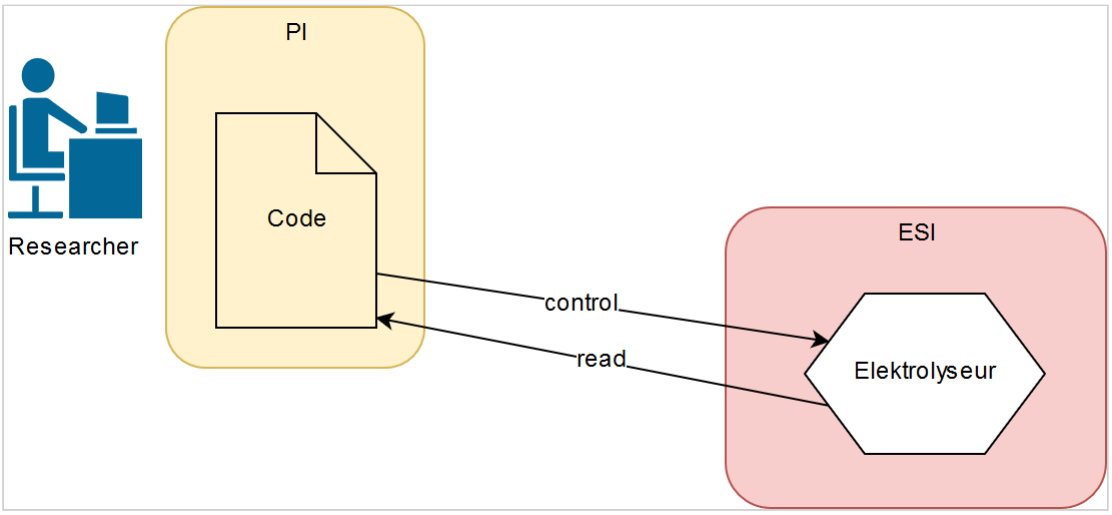
It must be possible to control Hardware at ehub and read their sensor values.



This must be possible with the following programming languages: Python, C, Matlab, LabView, Java, C#

1.1.2 Control at PSI

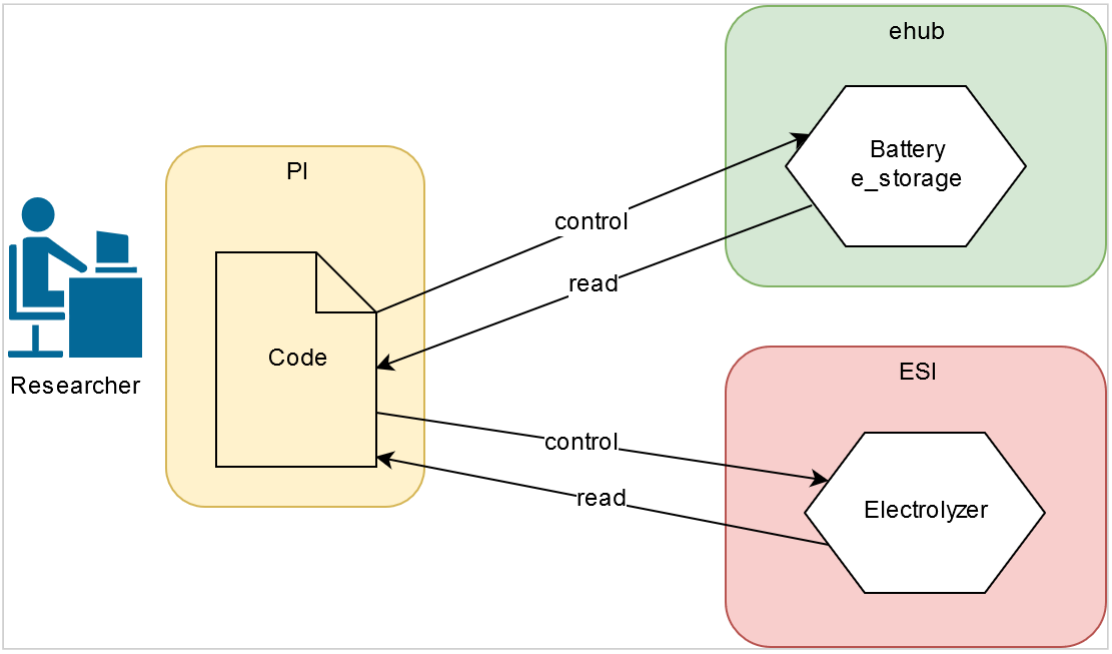
It must be possible to control Hardware at ESI and read their sensor values.



This must be possible with the following programming languages: Python, C, Matlab, LabView, Java, C#

1.1.3 Control at ehub and PSI at the same time

It must be possible to control Hardware at ehub and ESI and read their sensor values, from within the same code.



1.2 Archive

1.2.1 Persisting data stream ✓

The ReMaP platform receives sensor values from the hardware. These sensor values must be persisted, together with a timestamp (the timestamp can be created in the platform). A persisted sensor value with timestamp is called a measurement.

1.2.2 Persisting bulk data ✗

The ReMaP platform should be able to also persist complete measurements, that is sensor values that already have a timestamp. This data comes in bulks. These measurements usually have a high time resolution, where the measurement and timestamp are recorded close to the hardware, e. g. with hardware from National Instruments.

1.2.3 Accessing the archive ✓

The researcher must be able to access data from the archive (recent and past measurements) to process it further and analyze it. The data should be defined by time interval and hardware component (maybe also by a subset of measurement points).

Requirements for access restrictions to the archive are covered in the chapter "Rights Management".

1.2.4 Deleting data in the archive ✓

It must be possible to delete data in the archive.

1.2.5 Annotations for Experiments ✗

It would be nice to have annotations to the measurements, explaining which experiment it belongs to. ...TODO ...

1.2.6 Export of ehub-Archive to ReMaP ✗

It would be nice to do an export of all data from the ehub-Archive to the ReMaP archive.

1.2.7 Backup ✗

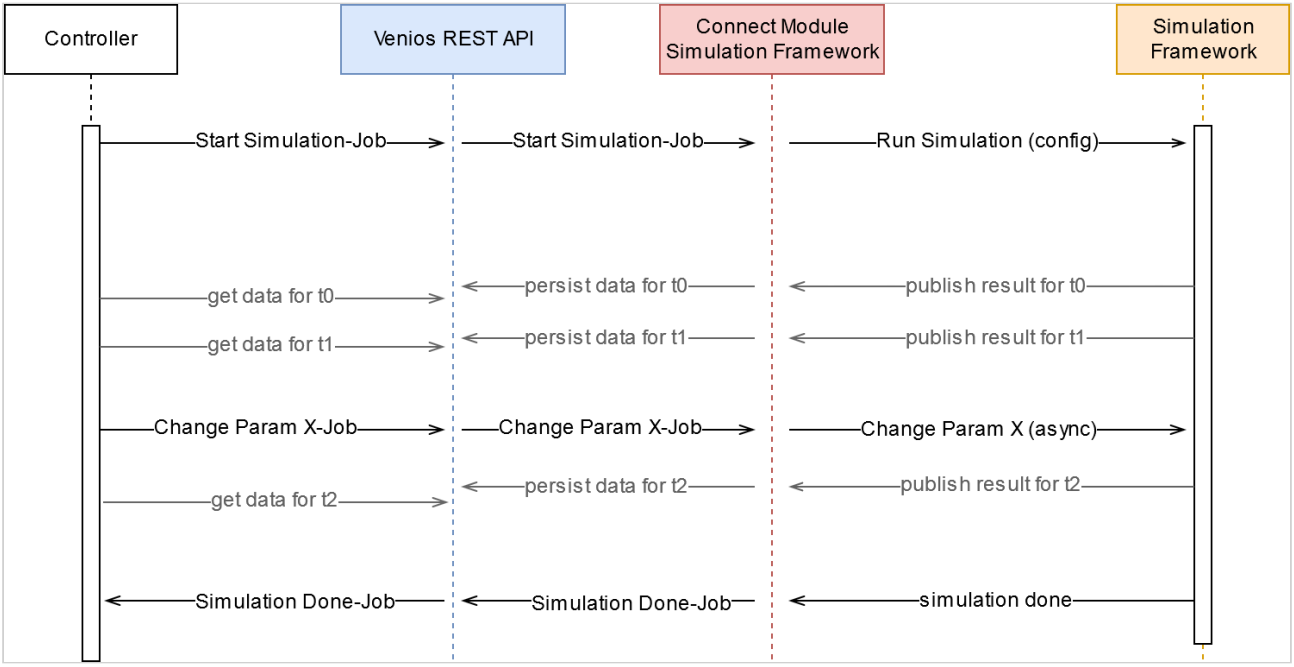
Regularly a backup of the data, specifically the measurements, is done and saved separately from the archive itself.

1.3 Simulation

1.3.1 Integration of Simulation Framework ✓

The simulation framework can

- be started over the ReMaP platform
- persist its results in the ReMaP platform
- get new parameters, while running
- return a "simulation done"-even



1.3.2 Simulation and Usage of access-restricted Archive Data

The following applies to any simulation (Simulation Framework, Adaptricity, City Energy Analyst):

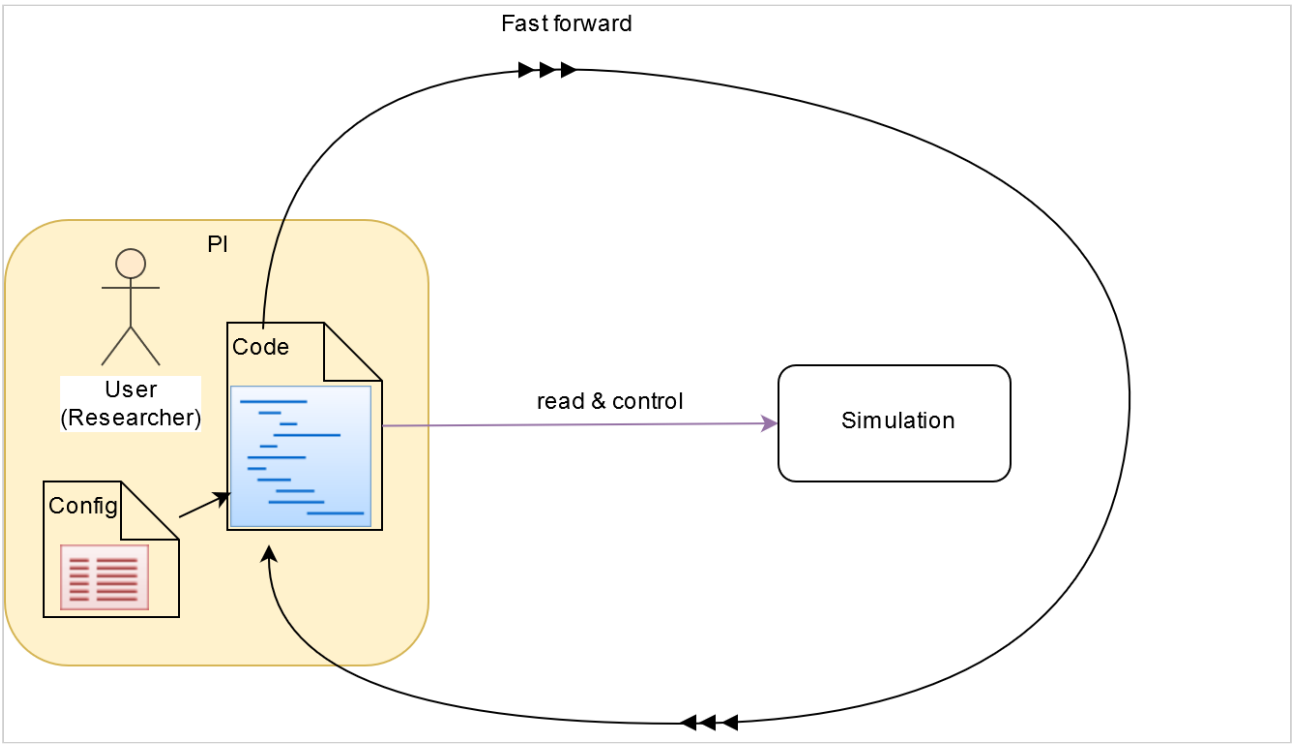
A simulation can either use archive data, that has no access restrictions. Or the archive data needs to be send by the user, that also uses the simulation (thereby preventing usage of data, that the user has no access rights for). If the user sends data with access restrictions to the simulation, he must use the Venios REST API. The user restricted data send to the simulation may not be used by the simulation for other purposes and may not be stored.

1.3.3 Testing Control Algorithm against Simulation

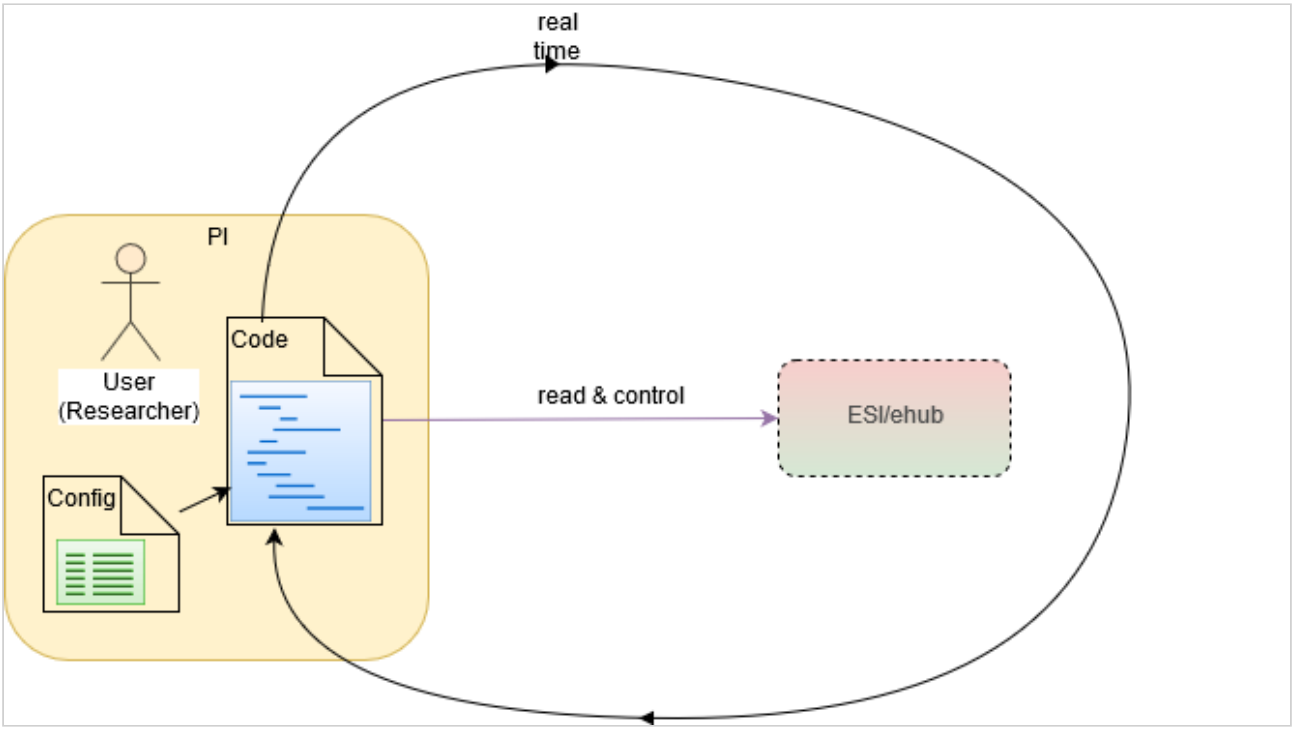
Before running the control algorithm on real hardware, it must be possible to run the same code against a simulation from the simulation framework. It must be possible to run the simulation faster than realtime.

The platform should provide a way to share a hardware simulation with other platform users. Depending on the simulation, there can be usage restrictions, that must be fullfilled.

1.3.3.1 First simulated



1.3.3.2 Then with real hardware



1.3.4 Scaled Mocks of real Hardware IN PROGRESS

It should be possible to create mocks of hardware, which are scaled versions of the real hardware (for example upscale a facility at PSI).

If this makes sense in ReMaP must still be discussed. Generally PSI is already doing scalings for CEDA. If additional scaling should be implemented on the ReMaP-platform still needs to be decided. PSI is also worried that the pure data from their facilities could be misinterpreted, when published on the platform.

1.4 Rights Management

> The requirements assume, that sensor values can only be accessed over the archive. Also note that all restrictions on "measurements" are at the same time restrictions on the archive.

1.4.1 Login

Only users which are registered for ReMaP are allowed to use any of the ReMaP functionality (i. e. control hardware over the platform or access data from the ReMaP-archive).

1.4.2 Synchronisation Users ehub ↔ ReMaP

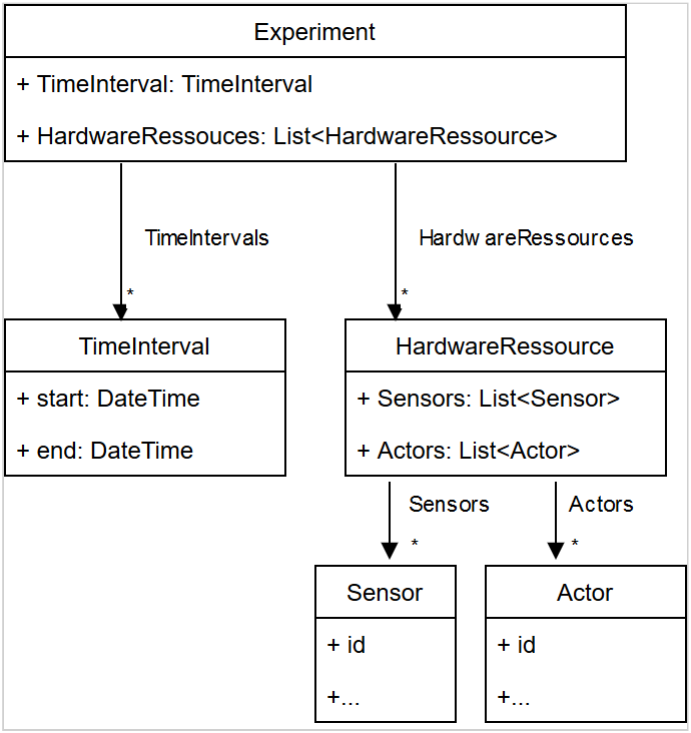
...Noch in Abklärung, was überhaupt machbar ist...

~~A user of ehub should (automatically, without manual work) also be a user of ReMaP. The user should be able to log in to ReMaP with the same credentials as for ehub.~~

~~When ReMaP connects to ehub, it should log in to ehub with the same user that is triggering the action in ReMaP.~~

1.4.3 Execution of an Experiment

For a ReMaP user, it must be possible to execute an experiment over ReMaP:



An experiment runs during one (or several) time intervals. It has one (or several) hardware resources.

A *hardware resource* consists of:

- *Sensors*, that publish sensor values to ReMaP. When a sensor value is persisted with a timestamp it is called a measurement.
- *Actors*, that provide control values to ReMaP-users. The user can control the hardware by setting control values of actors.

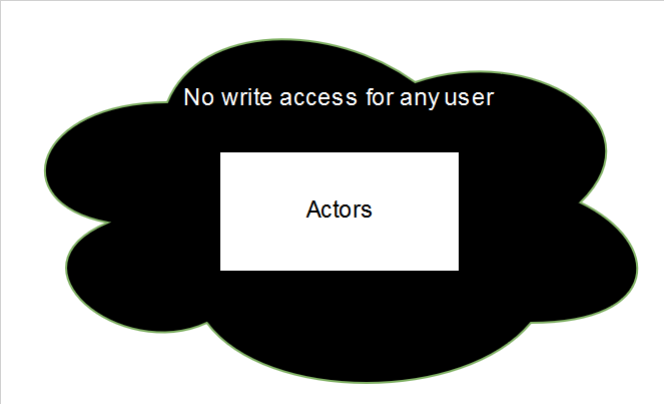
A sensor value, that is persisted with timestamp, is called a *measurement.* If sensor and timestamp are part of an experiment, we say it is a *measurement of the experiment*.

The user can access the current sensor value by reading the most recent measurement of a sensor.

To be able to execute an experiment the user must

- have read-access to the measurements of the experiment, during and after the experiment.
- have write-access to the actors of the experiment, during the experiment

1.4.4 No write access to Actors outside an Experiment

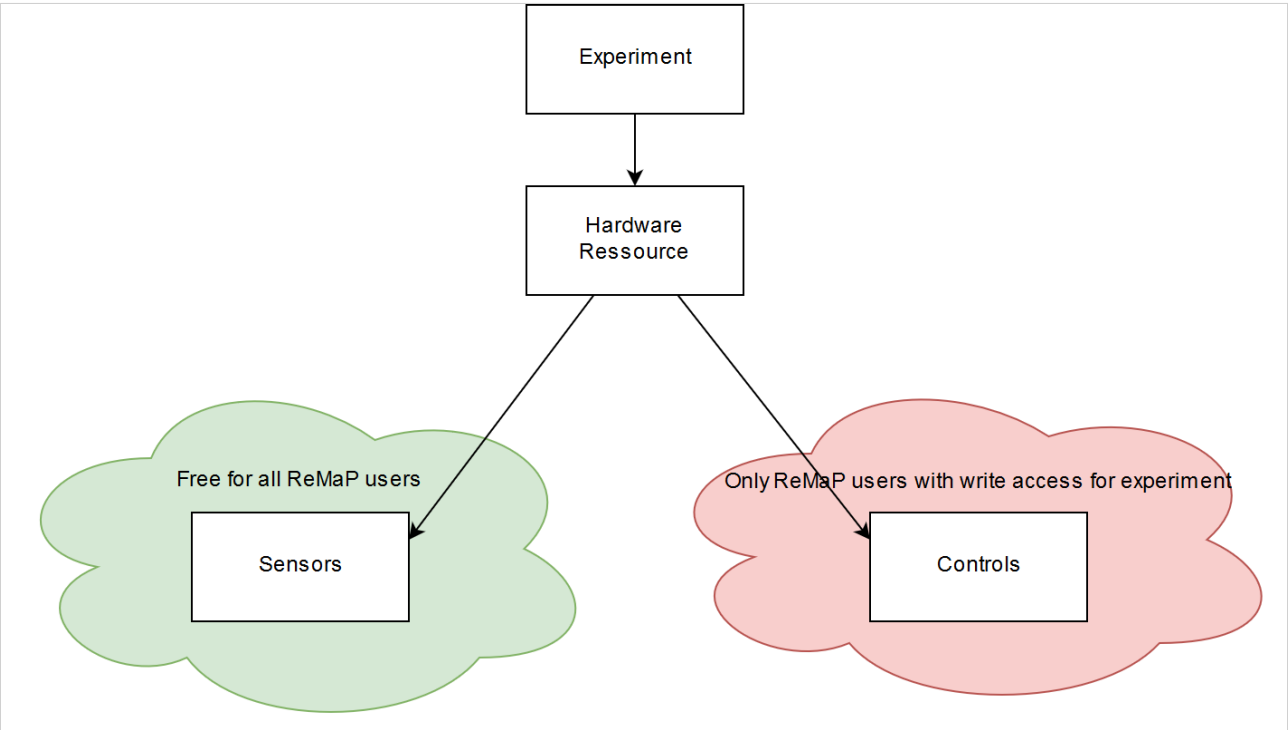


Per default, no user should have write access to the actors in ReMaP.

1.4.5 Read Access Restriction to Hardware Ressources outside an Experiment

For each hardware resource it must be possible to define, which users are allowed to have read-access on its measurements made outside any experiment.

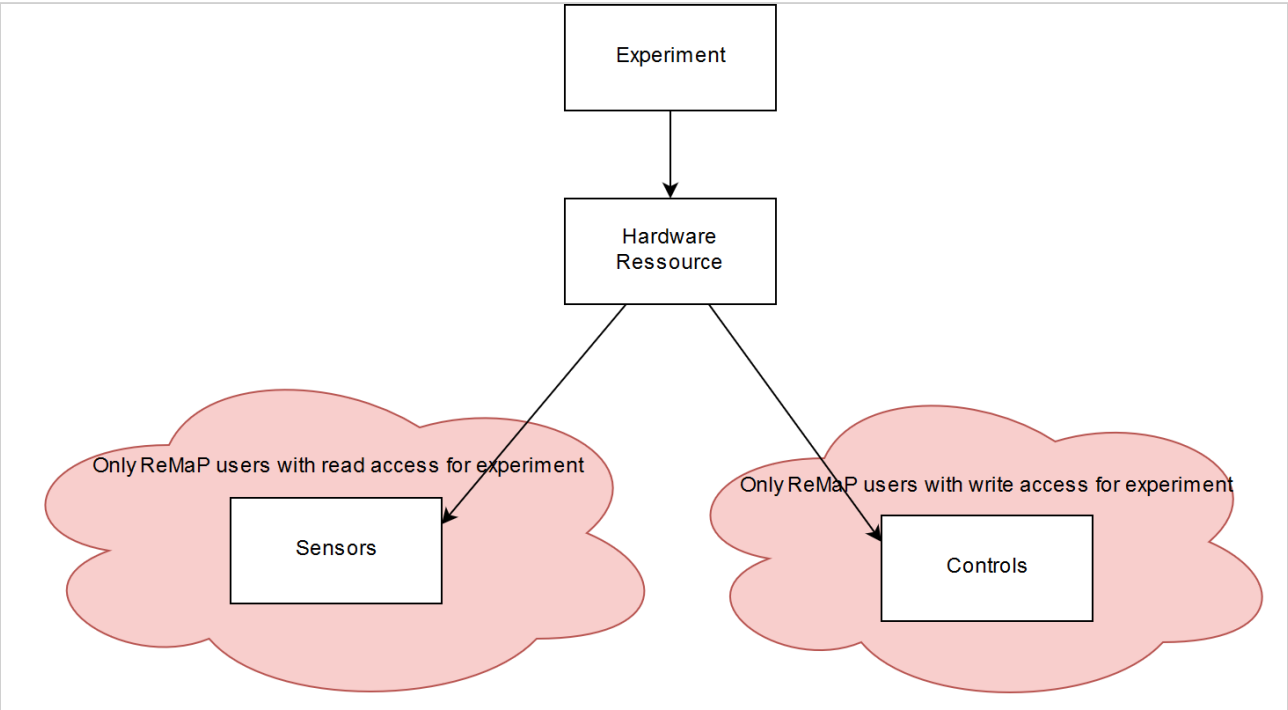
1.4.6 Full Access to Measurements of an Experiment



It must be possible to give all ReMaP-users read access to all measurements of an experiment.

This requirement is needed for EMPA.

1.4.7 Restricted Access to Measurements of an Experiment



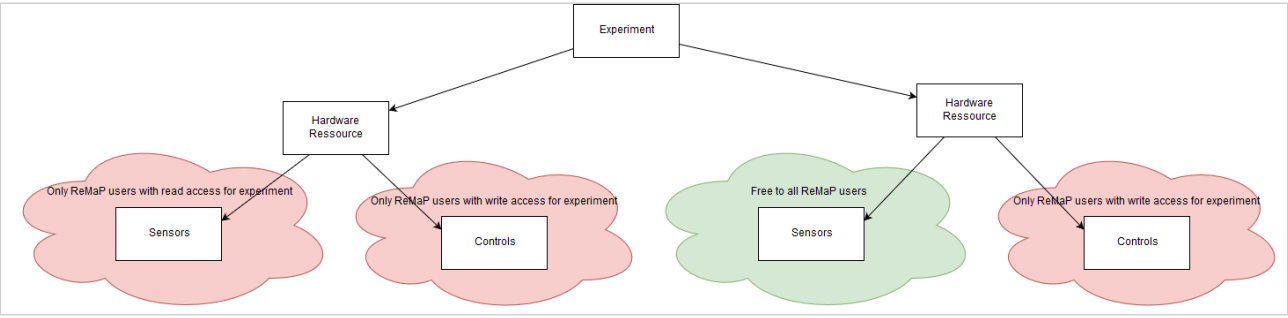
It must be possible to hide the measurements of an experiment from other ReMaP users.

This requirement is needed for PSI.

1.4.8 Grant read-access to an Experiment

It must be possible to give a user read-access, without write-access, to an experiment. This specifically means, that it is possible to give a user read-access to the archived measurements of an experiment.

1.4.9 Combination read access and read and write access per experiment



It must be possible to have a combination of the last two restrictions.

1.4.10 Full Read-Access to Subset of Measurements of an Experiment

After an experiment, it should be possible to give read-access to measurements of a subset of sensors to specific ReMaP users or all ReMaP users.

Requirement from PSI.

1.4.11 User Registration

A user can register his account (Username and Password) in Venios. A registered user has, per default, no rights in ReMaP.

1.4.12 Grant Access Right - Only Venios

A Venios-admin can set all possible access rights for all registered users.

There will be a process, involving ETH, EMPA, PSI on deciding who gets access to ReMaP and who gets what access rights. The Venios system administrator must conform to this process and only give access rights according to the defined process.

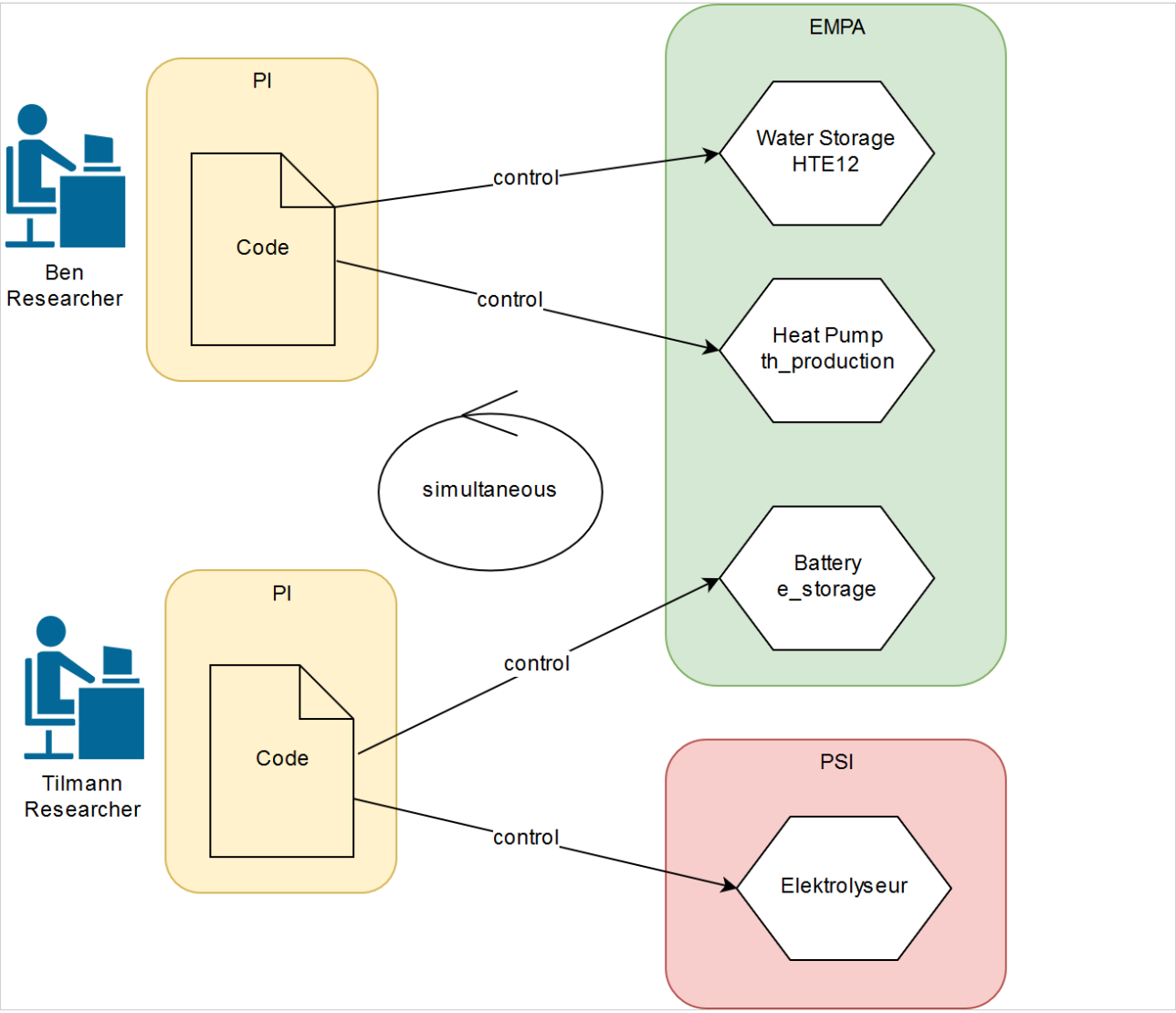
1.4.13 Grant Access Right - ReMaP Admin

ReMaP has an admin-UI, where a ReMaP-administrator can set all possible access rights for all registered users.

There will be a process, involving ETH, EMPA, PSI on deciding who gets access to ReMaP and who gets what access rights. The ReMaP system administrator must conform to this process and only give access rights according to the defined process.

1.4.14 Several Experiments running on different Hardware Resources at same time

It must be possible to run several experiments at the same time, controlling different hardware resources.

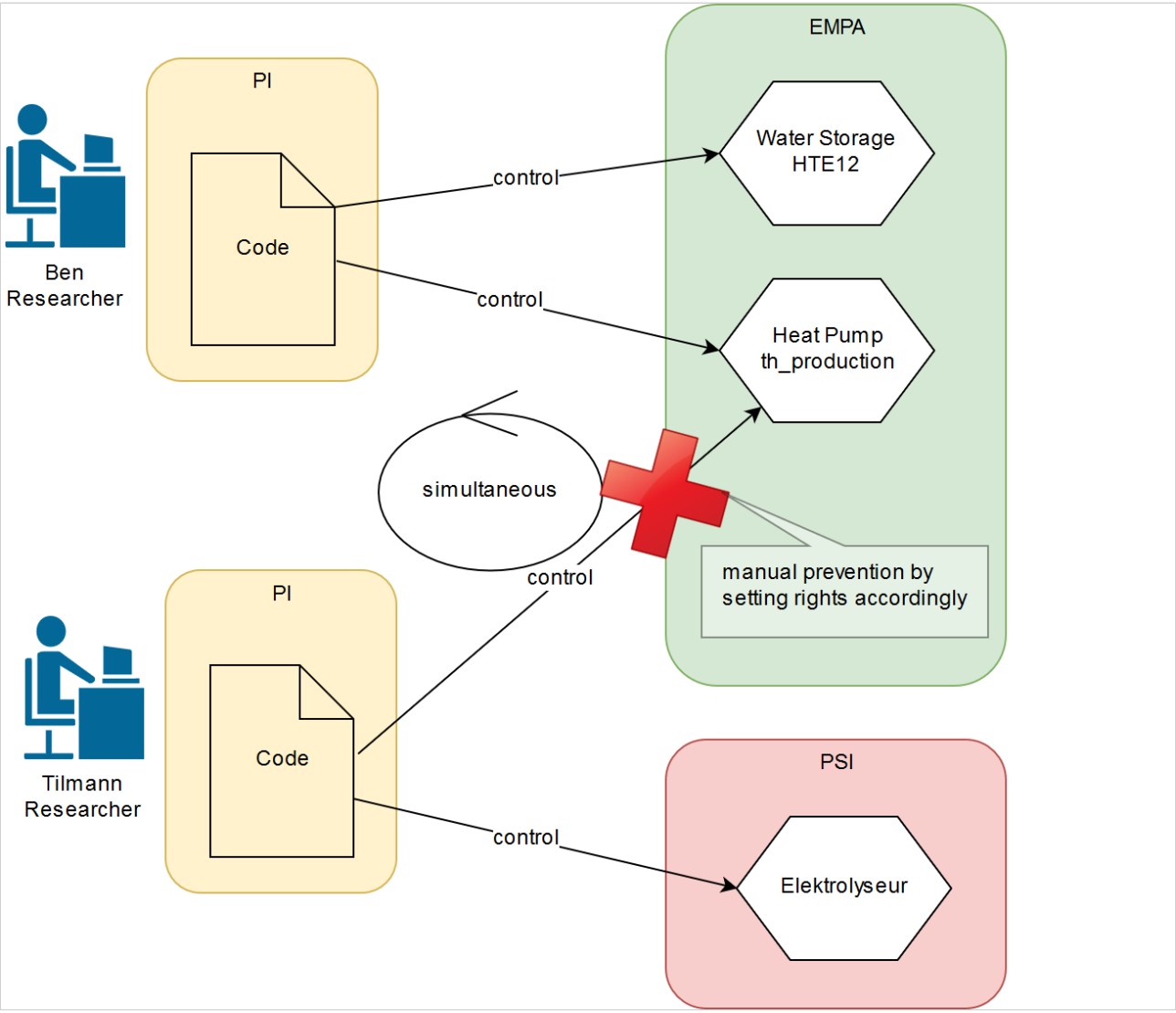


1.4.15 Manual Prevention of simultaneous Control of same Hardware Resource

It must be possible to prevent that two experiments with overlapping hardware resources run at the same time. The prevention is done by a manual configuration, e. g. by an admin setting user rights in the ReMaP platform accordingly.

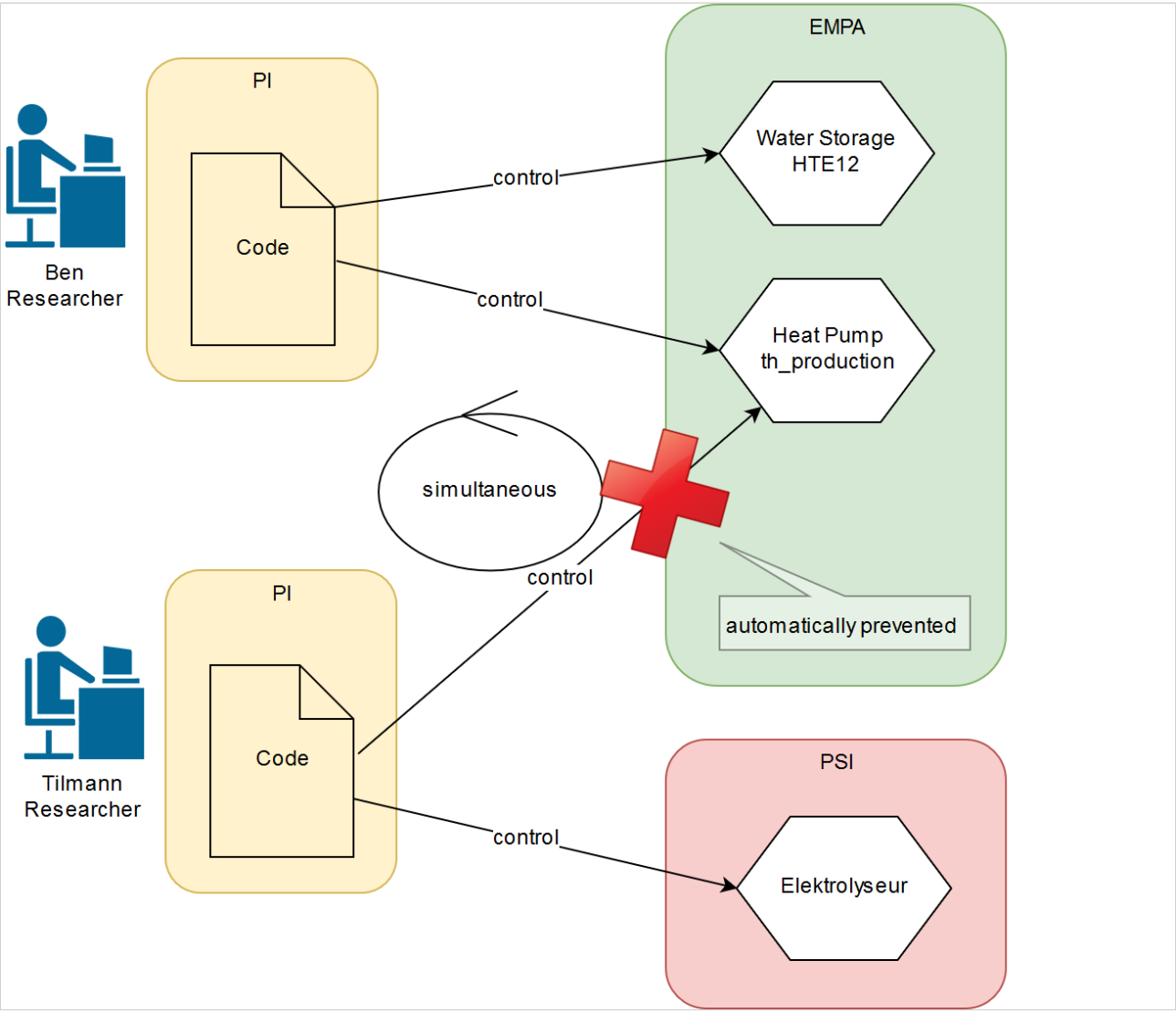
A mistake by the admin can lead to simultaneous control of the hardware. Therefore the admin must be reasonably trained and informed about the risks of running the plants. Additional safety measures could be taken, like the 4-eye-principle. The final, manual process needs to be defined and consolidated with PSI and EMPA.

Requirements in German by PSI: Der Admin muss über Gefahren durch den Betrieb von Anlagen instruiert worden sein. Er muss die notwendige Sorgfalt aufbringen, damit die Eintrittswahrscheinlichkeit solcher Fehler in einem akzeptierbaren Mass liegt. Gegebenenfalls auch mit weiteren Massnahmen, wie dem 4-Augen-Prinzip.



1.4.16 Complete Prevention of Simultaneous Experiments running on same Hardware Resource

It must be technically impossible that two experiments, with overlapping hardware resources, run at the same time.



1.4.17 Read-Access for non-ReMaP-Admins to current and future hardware control

It would be nice to have an overview

- on the planned experiments, specifically it should be visible: which users have write access to which hardware ressources at what time
- with read-access for non-ReMaP-admins

1.4.18 Scheduling

It would be nice to have a Web-UI, that helps scheduling an experiment. For example: The researcher enters into the UI, that he wants to execute an experiment. Then the UI would give an overview of possible free time slots.

1.5 Improvements for User Experience

1.5.1 Overview

It would be nice to have a Web-UI, that provides the following information

- What is the setup at EMPA and PSI? What Hardware do they have?
- What are the possible simulations, that the platform provides already?
- Tutorials on how to run my own control algorithm.
- Listing of the boundaries of the control values (e.g. lower and upper limit for a tank load).

1.5.2 Viewer

It would be nice to have a Web-UI that displays the current measurements for the researcher's running experiment. (Currently there is an application at EMPA that provides this functionality for the ehub)

1.5.3 Assistance for Coding ()

It would be nice, to have tools that help the researcher in writing code that interacts with the ReMaP platform. This could be in form of:

- Tool that generates example code to access the parts of the ReMaP API, that are interesting for the researcher (and that he has access to).
- Documentation and templates to use a tool like Soap UI, against which the researcher can debug his code

1.6 System Monitoring ()

A monitoring of system is necessary to provide a fast problem analysis. Additionally to being available for administrators, certain functionality should be accessible for researchers as well.

The monitoring should include:

For Administrators

- Logs of the ReMaP Software
- network connection
- Latency in network
- Health state of all ReMaP software components
    - ReMaP software in the platform
    - ReMaP software running in the institutes
    - Databases, Service Bus
- Health state of all VMs in the platform
- Connection state:
    - between ReMaP platform and ReMaP-connection-component at institute (with Venios this means between ConnectModule and Concentrator)
    - between ReMaP-connection-component and interface of institute (ehub/ESI) (with Venios this means between Concentrator and ehub/ESI)

For Researchers:

- Access to

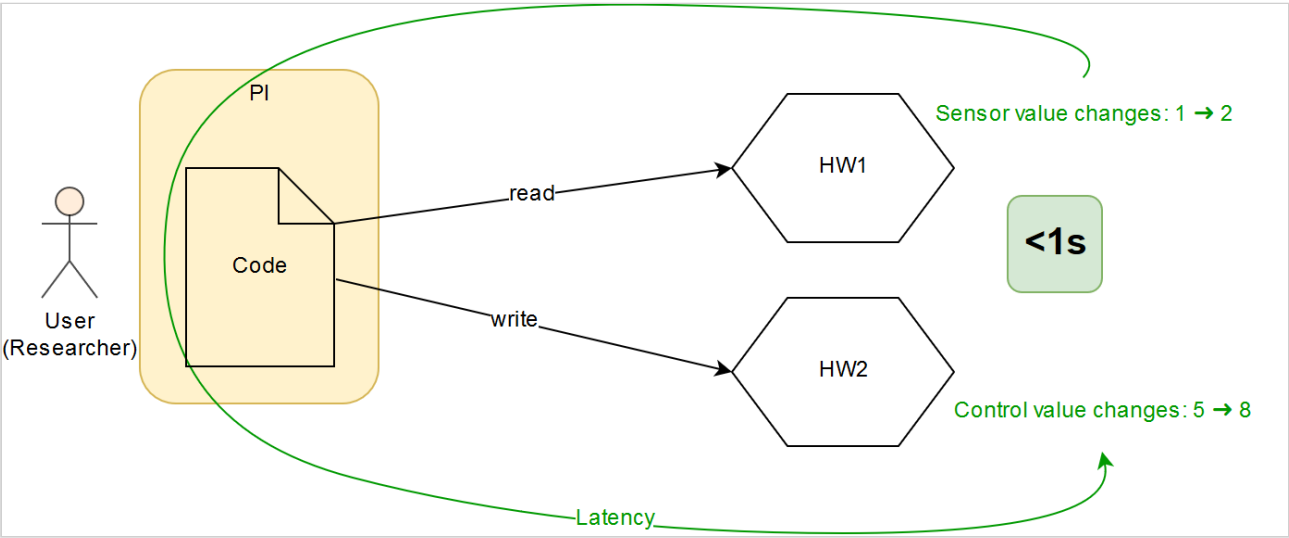
    - received measurements
    - sent control values
- Logs from the DUT (device under test, i. e. "the hardware")

2 Non-Functional Requirements

2.1 System Performance

2.1.1 Latency of 1 sec (✓)

The ReMaP platform must provide a roundtrip time of 1 second.



The time between a sensor value change (at the hardware) and the arrival of a new value (from the researcher's code at another hardware) must be 1 second or less.

Remark: There are systems at PSI where we will have a latency higher than 1 sec (but probably < 5sec). If this is a problem depends on the experiments.

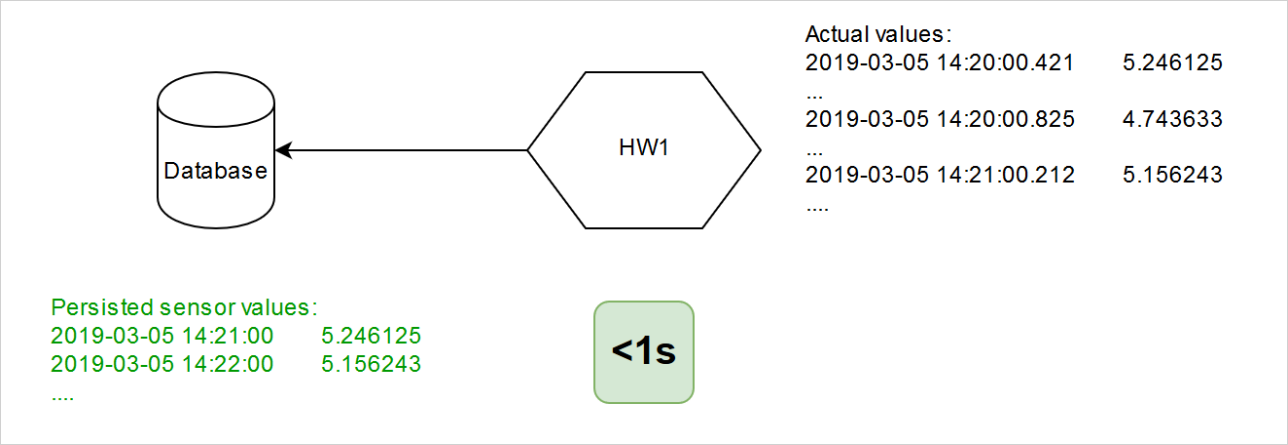
2.1.2 Latency < 1 sec ✗

Experiment T3.1 needs a latency of about 50-100ms.

Remark: This experiment will not run its fast control algorithms over the ReMaP platform.

2.1.3 Timestamp precision 1sec ✓

Regular measurements will get their timestamp in the ReMaP platform itself. The platform guarantees a precision of +/- 1 second.



2.1.4 Timestamp precision < 1sec

The following experiments need a timestamp precision lower 1 sec:

- T3.7 Power Electronics Testbench: Nanoseconds
- T3.1 Distributed State Estimation and System Operation:[1] 10-50ms, below 100ms.

2.2 Ease of extensibility

The following aspects should be relatively easy to add.

Minor changes in ehub or ESI should result in relatively low maintenance costs in ReMaP.

2.2.1 Add new hardware

It should be relatively easy to add new hardware to the system.

2.2.2 Add new simulations

It should be relatively easy to add new simulations to the system.

2.2.3 Add new experiments

It should be relatively easy to add new experiments (resp. let them run over ReMaP)

2.3 Security

The system must fulfill certain criteria concerning security:

- The requirements of the User Management must be ensured by state-of-the-art methods to secure websites.
- Connection to PSI and Empa must fulfill the requirements of their corresponding IT (This is a reminder, not a commitment to any new requirements stated by PSI or Empa).

1 https://tools.scs.ch/wiki/display/SGSREMAP/T3.1+Distributed+State+Estimation+and+System+Operation

- The answers to the issues of PSI: Offene_Punkte_ESI_ReMaP_V03.doc[2] must be fullfilled.
  Open issue: The question concerning, where the keys for the database encryption in Venios are stored is not yet settled, but must be settled for PSI to confirm these requirements.
- All data transfer over the internet must be authenticated and encrypted. This includes: transfer between the institutes and the platform, and transfer between researcher and platform.
- Direct database access is either restricted to ReMaP administrators or, if normal ReMaP users have direct database access, it must fulfill the rights given by the User Management

2 https://tools.scs.ch/wiki/download/attachments/68848047/Offene_Punkte_ESI_ReMaP_V03.doc?api=v2&modificationDate=1583504679781&version=1

3 System Architecture

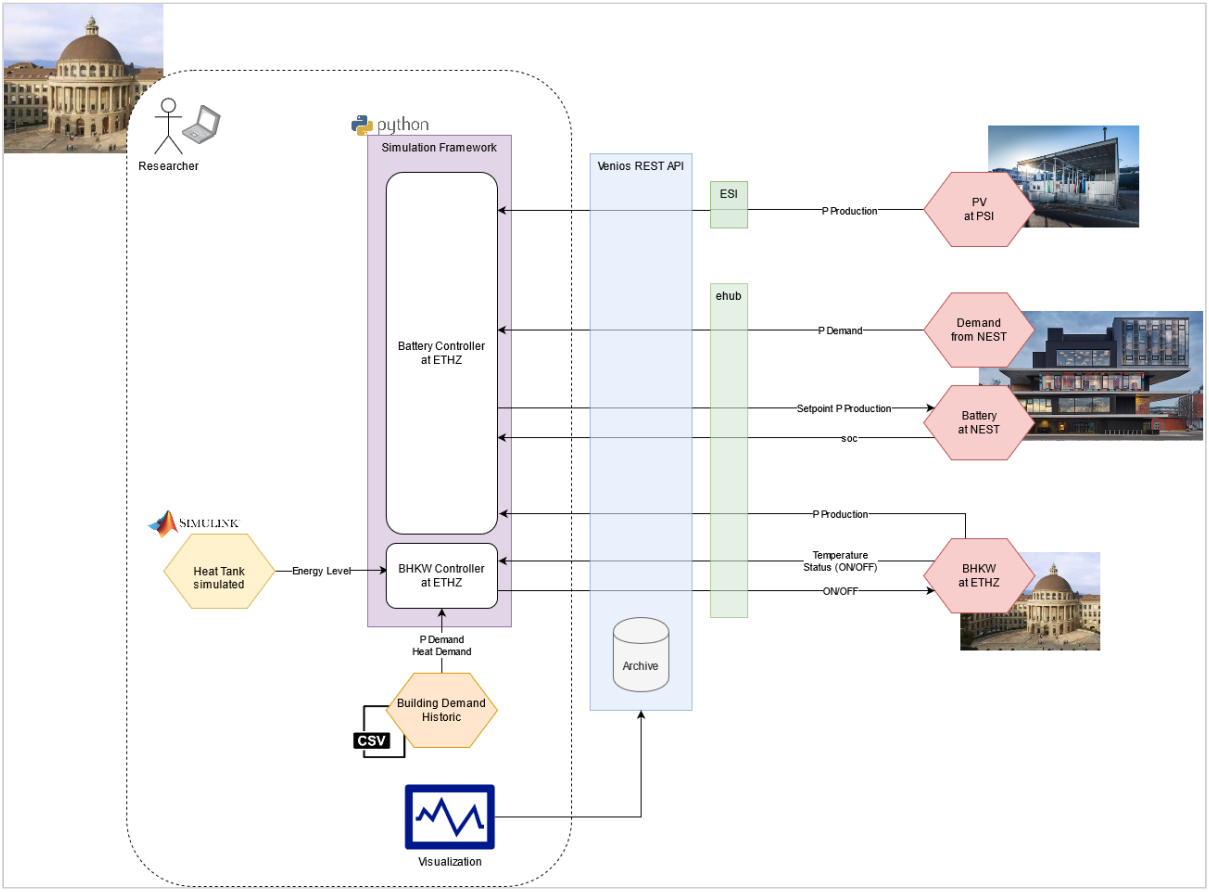
3.1 Introduction

For an overall idea of ReMaP, see the public website: https://remap.ch/platform/, for the requirements of the system see Functional Requirements(see page 7) and Non-Functional Requirements(see page 22)

Here we focus on the technical implementation and especially the parts provided by SGS (resp. Venios and SCS as subcontractors of SGS).

3.1.1 Example Implementation: Demo December 2020

The demo usecase, implemented for the ReMaP event in December 2020 gives a good overview on how ReMaP can be used:



Sorry, the widget is not supported in this export.
But you can reach it using the following URL:

https://vimeo.com/505629525

Starting from the outside, we see that hardware at PSI, NEST (Empa) and ETH are connected with simulations provided in Simulink and demand data provided as CSV-file. The control logic is implemented in Python and runs in the Simulation Framework. The researcher runs the Simulation Framework on his laptop and connects to the hardware (and possibly archive data) over the Venios REST API. Venios gives access (read and control) to the hardware of the ESI- and ehub-Platforms. Venios provides access to current and historic measurements, thereby providing data for any type of visualization.

The Simulation Framework (SFW) provides a framework to run an actual experiment. Each component (real and simulated) has a representing class in the SFW. The SFW connects the components and runs the control cycle of the experiment.

To connect to the hardware, the researcher needs to know the Venios-Id of the signal and its type, then he can use the custom interface in Python to connect to the signal.

The used signals (read, control and log) are provided in one file signals.py, in the following form:

```
read_signal_dummy_M00 = ReadSignal(
```

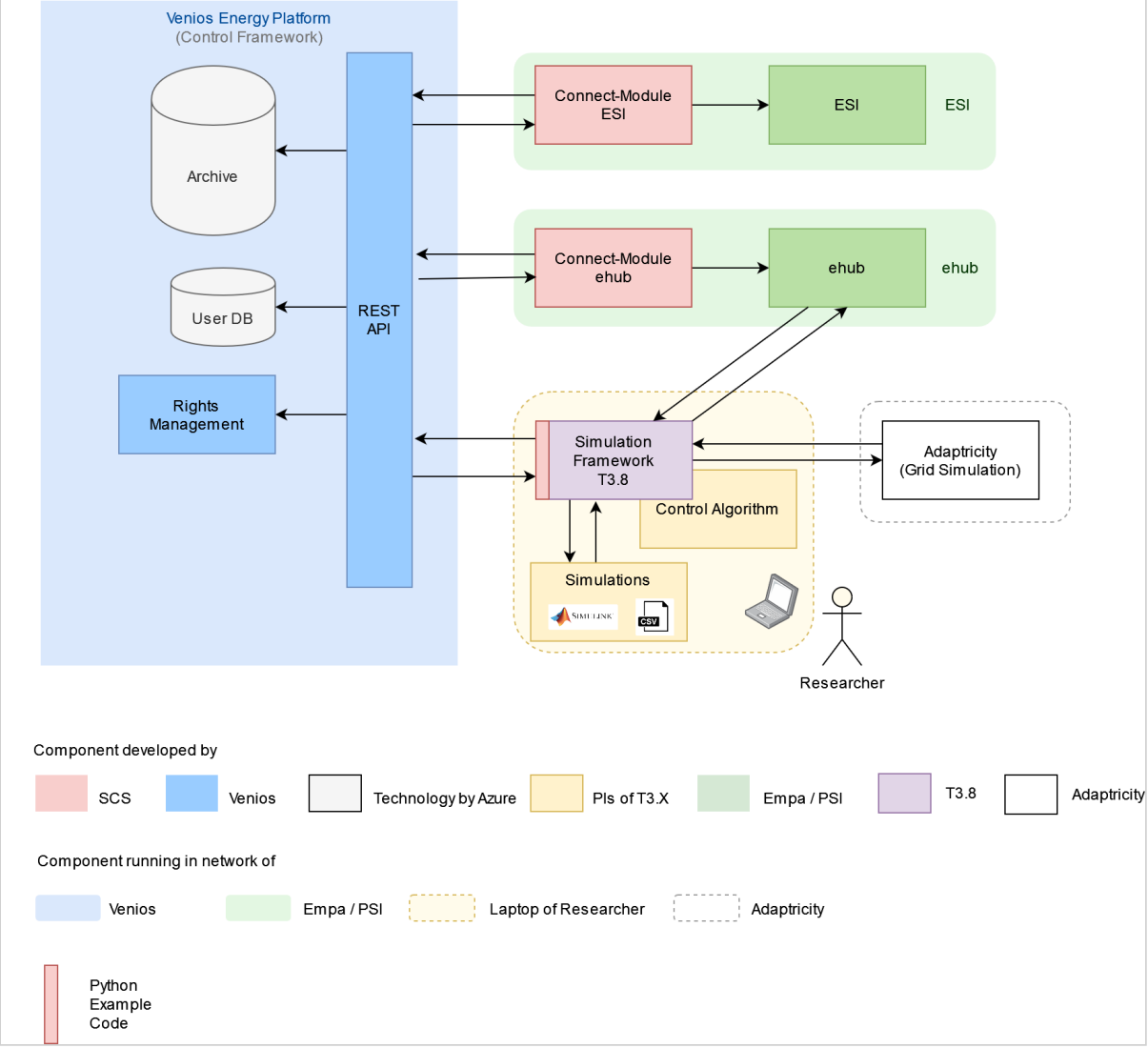
```
venios_id="ehub.65NT_DUMMY_Y720_M00",
classname_venios="FloatDataModel",
fieldname_venios="value",
field_type=float
)
```

Then, in the code the signal can be used together with the provided class VeniosApi, as here for reading a signal:

```
current_value = venios_api.read(read_signal_dummy_M00)
```

3.2 Overview ReMaP Components

The following gives an overview of the architecture of ReMaP:



The ehub- and ESI-platform are both connected to VEP by their own ConnectModule, running as a Windows Service in the ehub/ESI infrastructure.

The Simulation Framework is Python Code, that the researcher can run on his own laptop.

3.3 Usage of Venios Energy Platform (VEP)

The Venios Energy Platform (VEP) is used to provide access to the ehub- and ESI-platforms for researchers.

There is a stand-alone ReMaP-instance of VEP (ReMaP-VEP), with its own database and a custom API for ReMaP.

3.3.1 REST API

All communication with Venios is over its REST API (https://remap.venios.de:8282/index.html) and its WebSocket-interface.

3.3.2 Storage

As storage VEP uses NoSQL-Databases, specifically the following 2:

- Microsoft Table Storage[3]
- Microsoft Blob Storage[4]

3.3.2.1 Security

VEP persists the data using Azure Data Encryption-at-Rest[5]

> *Excerpt from* https://docs.microsoft.com/en-us/azure/security/azure-security-encryption-atrest
> What is encryption at rest?
> Encryption at Rest is the encoding (encryption) of data when it is persisted. The Encryption at Rest designs in Azure use symmetric encryption to encrypt and decrypt large amounts of data quickly according to a simple conceptual model:
>
> - A symmetric encryption key is used to encrypt data as it is written to storage.
> - The same encryption key is used to decrypt that data as it is readied for use in memory.
> - Data may be partitioned, and different keys may be used for each partition.
> - Keys must be stored in a secure location with identity-based access control and audit policies. Data encryption keys are often encrypted with asymmetric encryption to further limit access.
>
> In practice, key management and control scenarios, as well as scale and availability assurances, require additional constructs. Microsoft Azure Encryption at Rest concepts and components are described below.

3.3.3 Platform Access

- Login with username and password using OAuth
- All communication to REST API is using TLS
- Only communication over specific ports is allowed (can be configured).

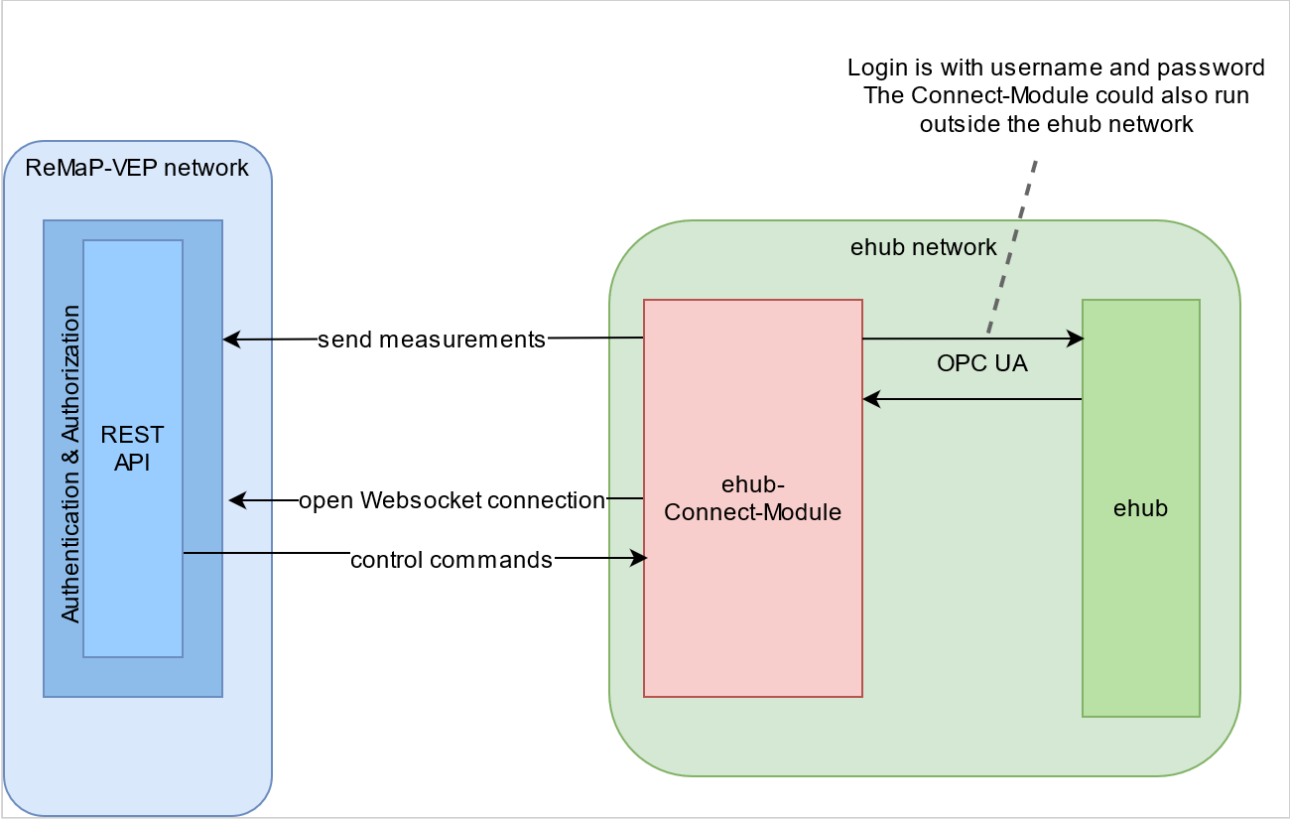
3 https://azure.microsoft.com/en-us/services/storage/tables/
4 https://azure.microsoft.com/en-us/services/storage/blobs/
5 https://docs.microsoft.com/en-us/azure/security/azure-security-encryption-atrest

- Security feature: Methods that don't exist, or that the user doesn't have rights to execute always return an empty reply. (This can be disabled by setting a special debug mode)

3.4 Connection to Hardware

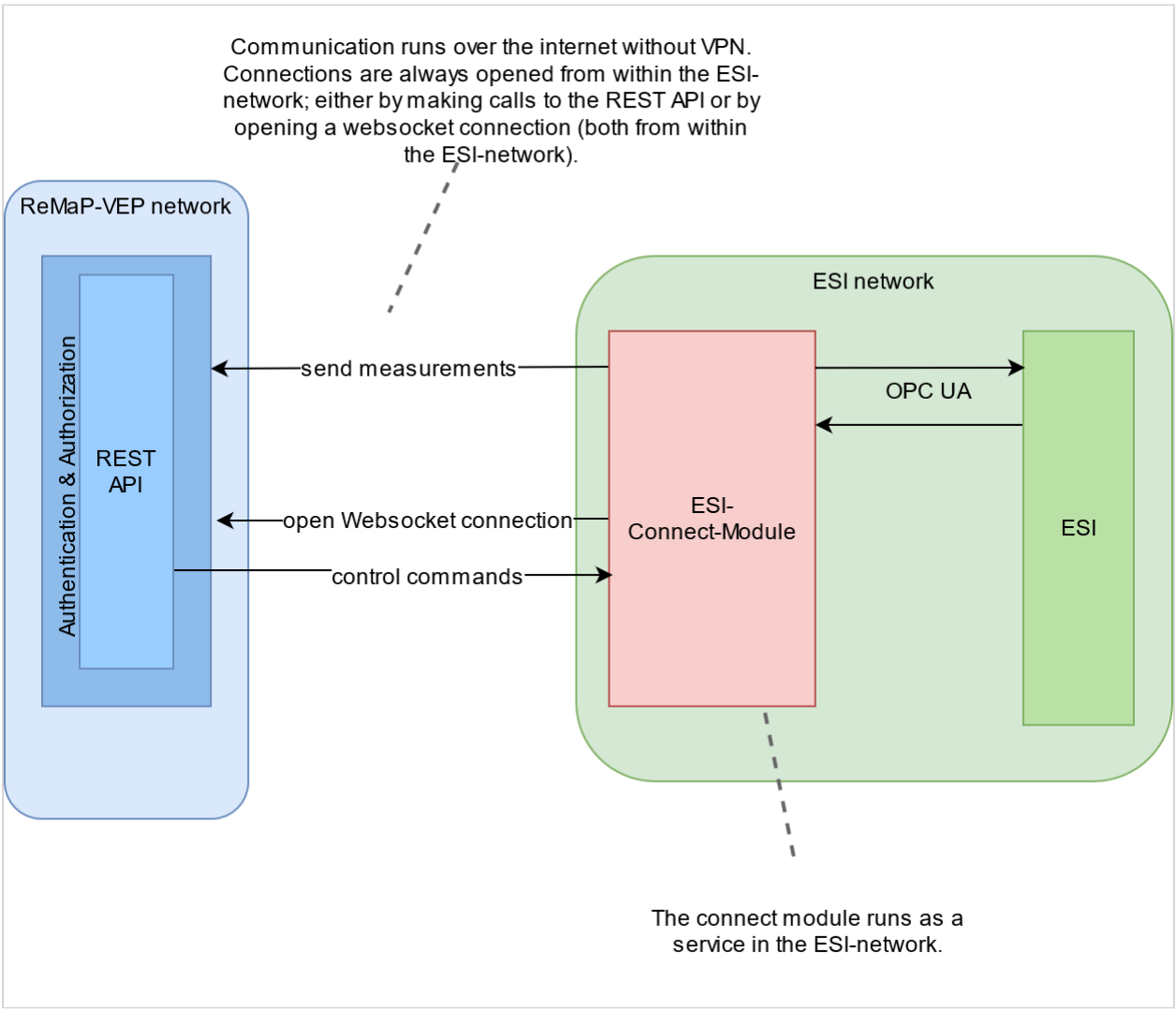
EMPA and PSI are connected to Venios by a proprietary Connect Modul, which connects the OPC UA-interface of ehub/ESI with VEP. Although both are using OPC UA, there is a proprietary protocol for ehub resp. ESI.

3.4.1 Connection to ehub



The ehub-Connect Module runs as a Windows Service in the ehub-network. It communicates with VEP over its REST API (using TLS encryption). It communicated with ehub over OPC UA, with username and password, using TLS. For ReMaP there is one "ReMaP"-user in ehub, that has the corresponding rights to access all components, that are integrated in ReMaP. The ehub-ConnectModule could also run outside the ehub-network.

3.4.2 Connection to ESI



The ESI-Connect Module runs as a Windows Service in the ESI-network. It communicates with VEP over its REST API (using TLS encryption). It communicated with ESI over OPC UA, with username and password, using TLS. For ReMaP there is one "ReMaP"-user in ESI, that has the corresponding rights to access all components, that are integrated in ReMaP. The ESI-ConnectModule could **not** be run outside the ESI-network.

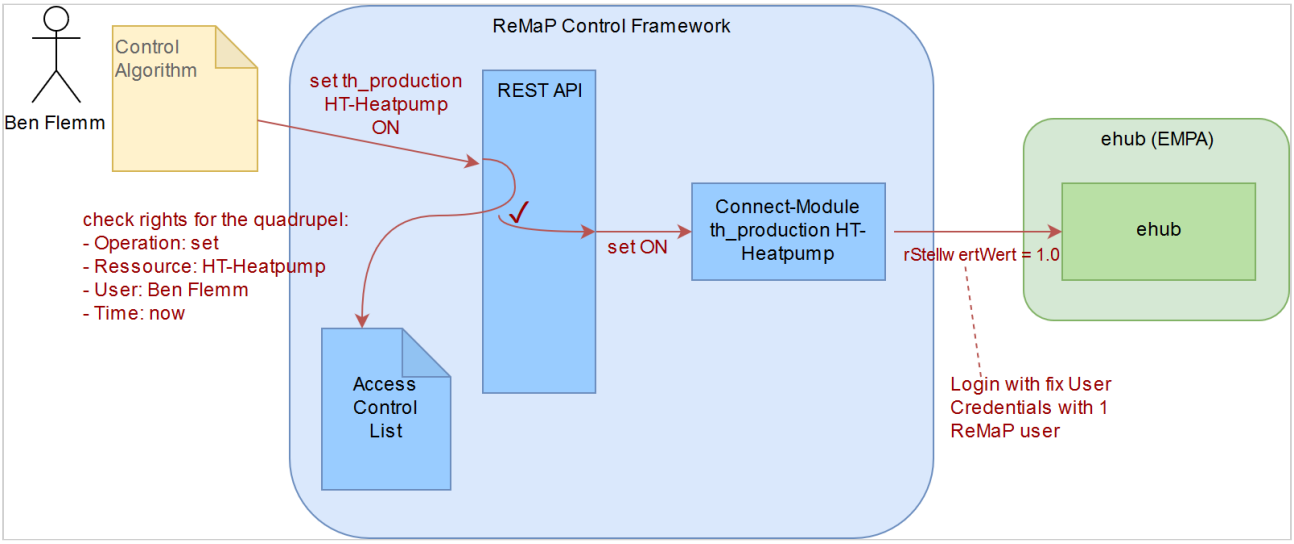
3.4.3 Parallel ConnectModules

For simplicity, there is currently one ConnectModule, running at ehub, and two ConnectModules running at PSI. In general, there can be many ConnectModules running in parallel, as long as they connect to different devices.

3.5 User Management

Venios provides a user management, fullfilling the requirements as described in Erfüllung Requirements(see page 49). Important to note is, that all restrictions are enforced by the REST API. A user cannot access data from the archive or execute an experiment, unless he has the corresponding rights, and the REST API gives him access to the according functionality. As all components communicate over the REST API, this is sufficient.

The following graph shows e. g. how control of hardware at EMPA is prevented:



3.6 Infrastructure

3.6.1 Cloud

The Venios Energy Platform runs on Azure Switzerland[6].

3.6.2 Deployment

New version of VEP are deployed by Venios.

ConnectModules: New versions are provided by SCS, here:

- Releases for Empa[7]
- Releases for PSI[8]

and deployed by Empa resp. PSI.

Python Example Code:

- Newest version available on github: https://github.com/ReMaP-CH/remap-controller-example

6 https://azure.microsoft.com/de-de/global-infrastructure/switzerland/
7 https://tools.scs.ch/wiki/display/SGSREMAP/Releases+for+Empa
8 https://tools.scs.ch/wiki/display/SGSREMAP/Releases+for+PSI

3.7 Usage of ReMaP-VEP

3.7.1 Connecting to ReMaP-VEP

3.7.1.1 REST API

The functionality of ReMaP-VEP is provided by a REST API. With this any high level programming language (like Python, Java, C, ...) can be used to access ReMaP-VEP.

A quick start on how to use it, is found here: Basic Venios Access via Browser[9]

For more advanced usage, the Python Example Code and its documentation[10]should be looked at.

3.7.1.2 Login

The user has to provide valid username and password to access ReMaP-VEP. ReMaP-VEP has its own user database. New users are created by Venios. Access rights are configured by SCS.

Technologies for the login/communication/session management are OAuth with TLS.

3.7.2 Basic assistance

3.7.2.1 Example Code

SCS provided example code in Python[11] to access the ReMaP REST API. The examples include

- correctly login to the platform, with OAuth Tokens
- setting of control value
- read measurement values
- setting up polling mechanism to check for changing sensor value

3.8 Example: Data flow for a simple Control Algorithm

The following gives an illustrative example on the flow of information through the whole ReMaP system.

The usecase is: A control algorithm continuously checks the PV production at PSI. Depending on the value, the setpoint for the energy production of a battery at NEST is set.

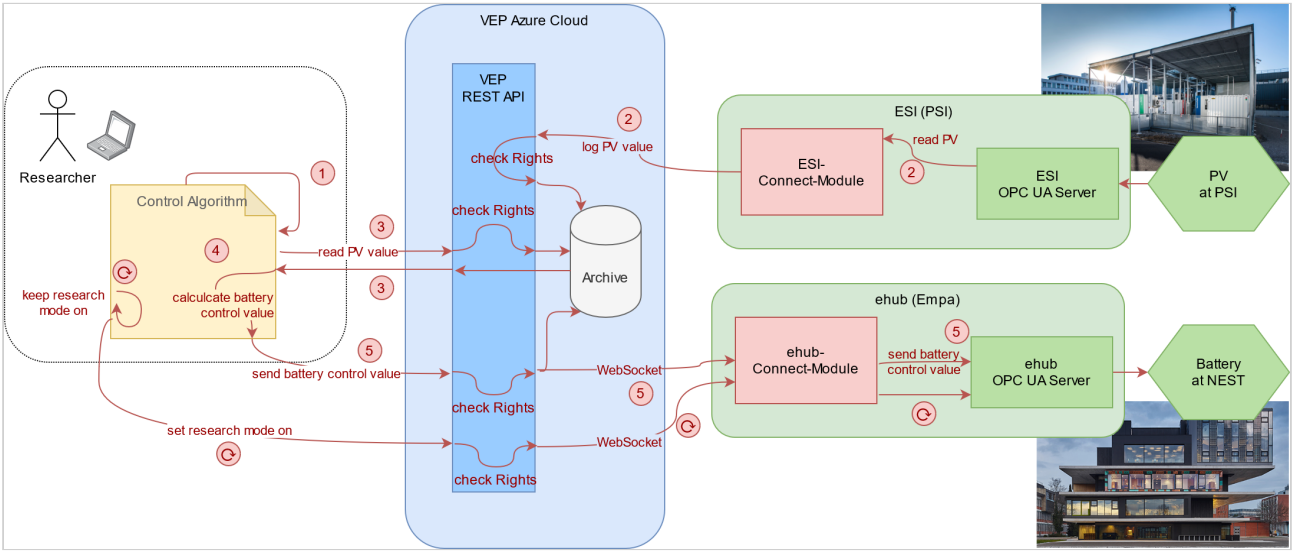
1. Before the control logic starts, the communication with the ReMaP system has to be setup, this includes:
    a. Checking if all read-signals have current values
    b. Requesting and waiting for control access to the battery at NEST
2. The PV production is continuously persisted in VEP
3. The control algorithm continuously checks the most recent value of the PV production
4. The control algorithm decides, if a new control value should be send to the battery at NEST

9 https://tools.scs.ch/wiki/display/SGSREMAP/Basic+Venios+Access+via+Browser
10 https://tools.scs.ch/wiki/display/SGSREMAP/Python+Example+Code
11 https://tools.scs.ch/wiki/display/SGSREMAP/Python+Example+Code

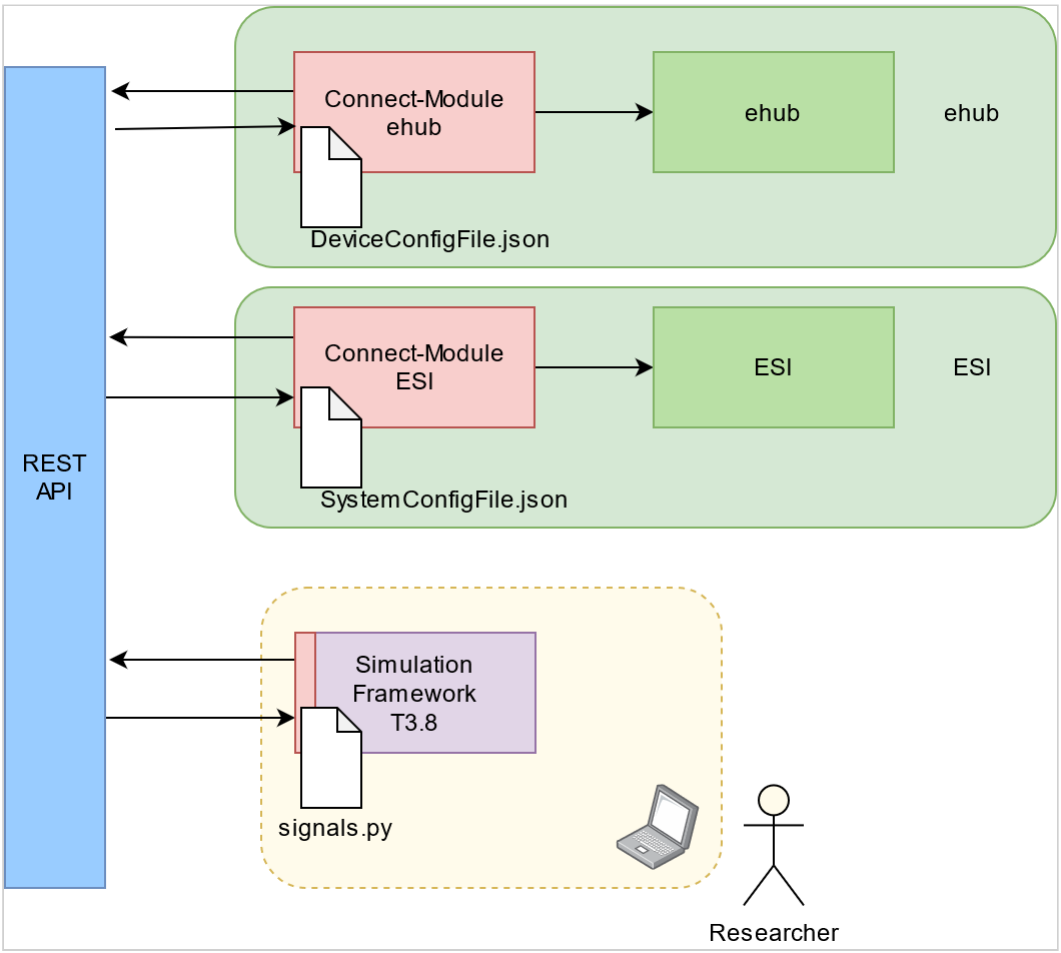
5. A new control value is send to VEP.
   The ehub-ConnectModule receives the new control value over a WebSocket connection from VEP and forwards it to the OPC UA Server of ehub.
   The ehub OPC UA Server forwards the control value to the battery.

⟳ The control algorithm continuously sends a "set research mode on"-command to VEP.
This triggers the ehub-ConnectModule to send the corresponding watchdog-commands to ehub.
This way the researcher stays in control of the battery throughout the experiment.

Rights management: VEP makes sure that researcher and the ConnectModules have the corresponding rights needed. For example, the PV signal from PSI can only be archived by the ESI-ConnectModule, but not by the researcher and also not by the ehub-ConnectModule.



3.9 Integration of External Systems



The hardware at Empa and PSI is both integrated in Venios with their corresponding ConnectModules. The ConnectModules are written in C# and both run as a Windows Service in the infrastructure of the corresponding institute.

To connect new hardware from ehub/ESI the config-file

- ehub: DeviceConfigFile.json[12]
- ESI: EsiSystemConfigFile.json[13]

has to be adjusted and the ConnectModule has to be restarted.

The configuration files, can be generated from CSV-files. How this is done, is documented here:

- ehub Signallist Translator[14]
- ESI Signallist Translator[15]

The configurations particularly contain the definition of the signals.

12 https://tools.scs.ch/wiki/download/attachments/126527458/DeviceConfigFile.json?api=v2&modificationDate=1629987323944&version=1
13 https://tools.scs.ch/wiki/download/attachments/126527458/EsiSystemConfigFile.json?api=v2&modificationDate=1629987363141&version=1
14 https://tools.scs.ch/wiki/display/SGSREMAP/ehub+Signallist+Translator
15 https://tools.scs.ch/wiki/display/SGSREMAP/ESI+Signallist+Translator

Example Signals ehub:

VeniosId	ClassnameVenios	FieldnameVenios
ehub.65NT_DUMMY_Y720_M00	FloatDataModel	value
ehub.65NT_DUMMY_Y720.ControlStatus	EHubStatusModel	researchModeConfirmation
ehub.65NT_DUMMY_Y720.ControlStatus	EHubStatusModel	researchModeApproval
ehub.65NT_DUMMY_Y720.ControlStatus	EHubStatusModel	researchModeStatus
ehub.65NT_DUMMY_Y720.ControlStatus	EHubStatusModel	alarmActive
ehub.65NT_DUMMY_Y720.ControlStatus	EHubStatusModel	alarmStatus
ehub.65NT_DUMMY_Y720.ResearchAccess		
ehub.65NT_DUMMY_Y720.ResearchAccess		
ehub.65NT_DUMMY_Y720_Y00	FloatControlModel	value

Signalliste_ehubDummy.csv[16]

Example Signals ESI:

TeilsystemId	VeniosId	ClassnameVenios	FieldnameVenios
ESI.Dummy	ESI.Dummy.Status	PSIStatusModel	realDataApproved
ESI.Dummy	ESI.Dummy.Status	PSIStatusModel	cedaDataApproved
ESI.Dummy	ESI.Dummy.Status	PSIStatusModel	controlApproved
ESI.Dummy	ESI.Dummy.9ESI01DU005.Control	FloatControlModel	value
ESI.Dummy	ESI.Dummy.9ESI01DU001	FloatDataModel	value
ESI.Dummy	ESI.Dummy.9ESI01DU002	FloatDataModel	value
ESI.Dummy	ESI.Dummy.9ESI01DU003	BoolStatusModel	value
ESI.Dummy	ESI.Dummy.9ESI01DU105	FloatDataModel	value

Signalliste_esiDummy.csv[17]

The researcher can connect to the signals, by configuring a signallist in Python:

example_signals.py

```
read_signal_dummy_M00 = ReadSignal(
    venios_id="ehub.65NT_DUMMY_Y720_M00",
    classname_venios="FloatDataModel",
    fieldname_venios="value",
    field_type=float
)
```

example_signals.py[18]

The defined signals can then be used together with the VeniosApi[19]. For example, we can read the most current value of a signal, with the following line:

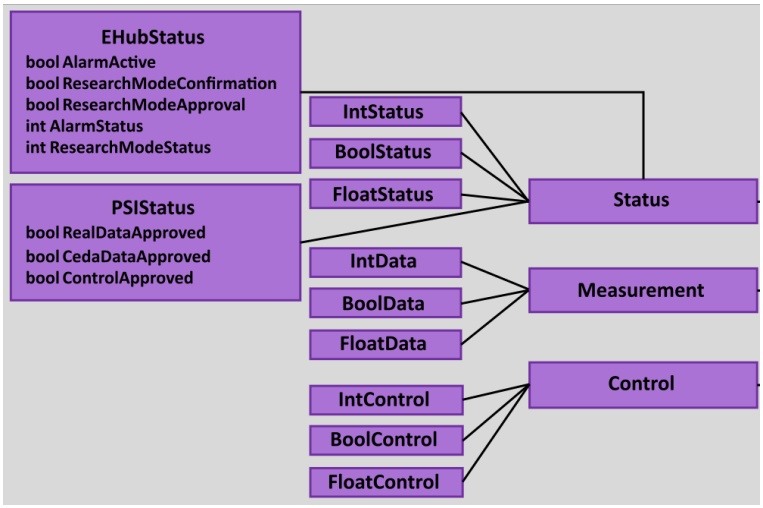
16 https://tools.scs.ch/wiki/download/attachments/126527458/Signalliste_ehubDummy.csv?api=v2&modificationDate=1630420828784&version=2
17 https://tools.scs.ch/wiki/download/attachments/126527458/Signalliste_esiDummy.csv?api=v2&modificationDate=1630420852054&version=2
18 https://tools.scs.ch/wiki/download/attachments/126527458/example_signals.py?api=v2&modificationDate=1630054901906&version=1
19 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/venios_api.py

```
current_value = venios_api.read(read_signal_dummy_M00)
```

A detailed documentation on how to use the VeniosApi can be found here: Python Example Code[20]

3.9.1 Datamodel

To have full flexibility between systems, and to make it easy to connect to new signals, the datamodel is based on single signals. Each signal has a VeniosId (e.g., ehub.65NT.EXP20.T200.ElectricalStorage.BidirectionalInverter.ActivePowerTotal) and a type (e. g. float). To group signals, the VeniosId is used as a descriptive name, providing a hierarchy as to where the signal is located.



3.9.1.1 Status, Measurement

Status and Measurement can be used interchangeable. Currently there is no difference in the handling of them.

3.9.1.2 Control

The Control-Classes are used for control signals. **TODO: Link to explanation on Control**

3.9.1.3 EHubStatus / PSIStatus

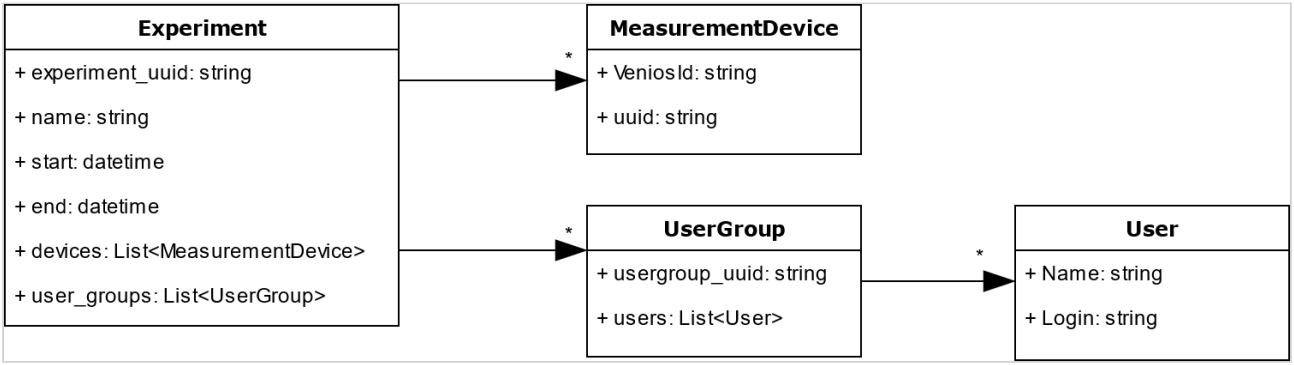
EHubStatus and PSIStatus are the only exception to having "simple signals". The signals connected to EHubStatus and PSIStatus have special meaning. With these it can be checked, if control access is given.

3.9.2 Access Restriction

3.9.2.1 Experiment

In order for a researcher to access any signals, an "Experiment" needs to be defined in Venios:

20 https://tools.scs.ch/wiki/display/SGSREMAP/Python+Example+Code



The experiment has a time interval (start, end), a list of devices and a list of users (within different usergroups). The users can access any measurements from the devices, that have a timestamp within the time interval (also when the timestamp is in the past). Control access to control-devices is only given within the time interval. So once the time interval is completely in the past, no user can control the devices of the experiment anymore.

Experiment UUID

A researcher needs to provide the experiment_uuid of his experiment. This is done at the top of the signals.py file:

signals.py

```
experiment_uuid = '4fedcc94-0a6e-11ec-9a03-0242ac130003'
```

The experiment_uuid is provided by the ReMaP administrator, currently by SCS, Sabine Proll.

The currently defined experiments can be accessed by any ReMaP-user, as described here: Current Access Rights[21]

3.9.2.2 Logging Measurements

Logging measurements is independent of the configured experiments. Generally only one user can log measurements for a signal. This is the same user, that also created the corresponding MeasurementDevice.

For example: when the ConnectModule is first started, it checks if any devices are missing and then creates them in Venios. After that, the ConnectModule has the right to log its measurements. When the researcher logs (simulated) measurements, this also works. But only the user who created the devices in the beginning can log measurements. This right cannot be passed to another user.

3.9.3 Sampling Rate

Signals are sampled corresponding to the configured SampleIntervals in the appsettings.json of the ConnectModule:

21 https://tools.scs.ch/wiki/display/SGSREMAP/Current+Access+Rights

appsettings.json

```
"PeriodicSampleInterval": 300000,
"OnChangeSampleInterval": 2000
```

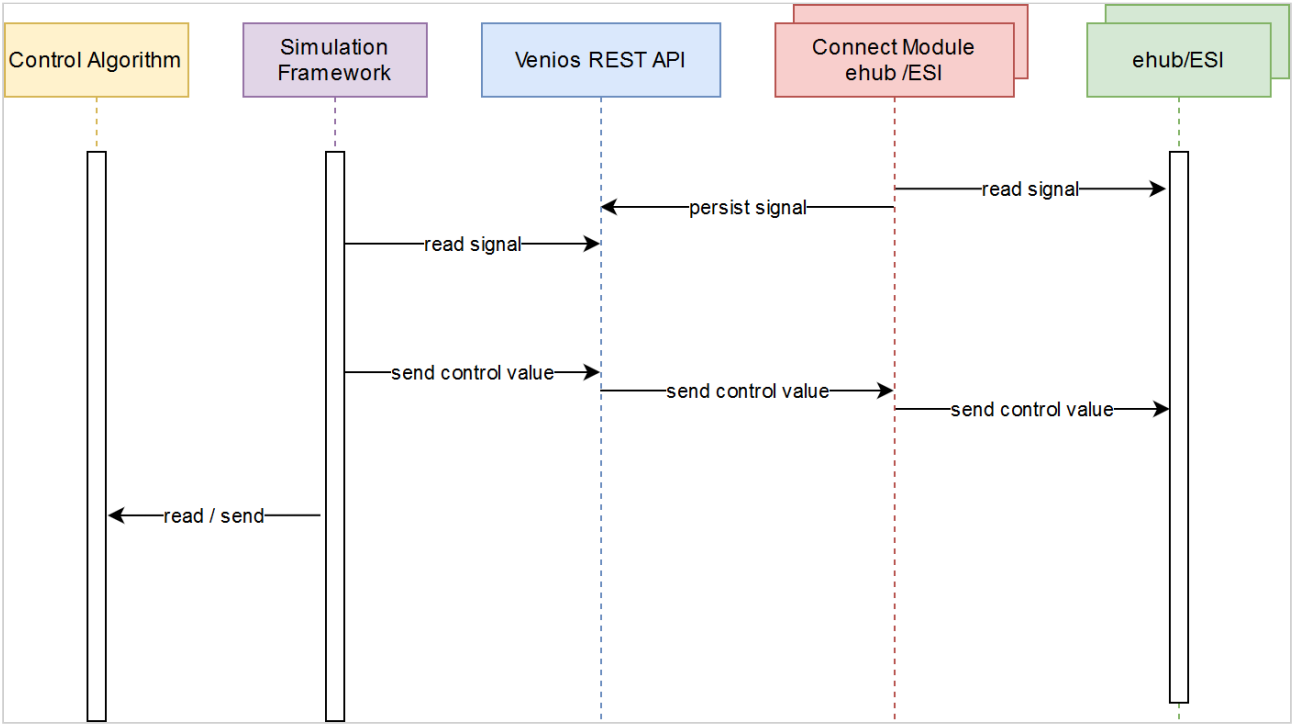
	Read Signals (Status, Measurement)	Control Signals
PeriodicSampleInterval in ms	Read and Control Signals are always sampled at this rate. Also, when the value does not change.	
OnChangeSampleInterval in ms	Read signals are sampled, whenever they change, but not faster than OnChangeSampleInterval	Ignored. Control signals are always sampled when they change. Independent of the frequency of change.

3.9.4 Open Issues

- Budget needs to be checked:
  (Theses improvements would be very desirable, but the experiments can run without them. Therefore the decision on their implementation is currently postponed.)
  - Units
    The current idea, that units are integrated in Venios and specified in the signallists, and when reading (and writing) signals to and from the Venios REST-API.
  - Enums/Status-Information:
    Ideally for each integer-signal, that acts like an enumeration (as typical with status information, as e. g. 0 - on, 1 - starting, 2- running, 3 - stopping, 4 - error), there is information on the meaning of each integer. This information should be provided by Empa and PSI with the signallist and it should be accessible by the researcher through the Venios REST API.
- Further open questions:
  - Datamodel / interchangeability between systems: When someone (e. g. a researcher) maps signals of two systems (e. g. two different batteries) to each other, where is this defined? And how can it be reused by the next researcher?
  - General information on the hardware: How is general information on the hardware provided/stored?

3.9.5 Integration Simulation Framework

3.9.5.1 Interaction between Venios and Simulation Framework



The simulation framework can

- interact with real hardware over the CFW

3.9.5.2 Features provided for SFW

See Tasklist SCS (Integration SFW)(see page 40)

3.9.5.3 Access restriction for Archive Data

The SFW will only store its own data as archive data, not data retrieved from ReMaP hardware. It will access archive data, with the login-credentials provided by the user of the SFW. It is within the responsibility of the user to not use the archive data in the wrong way.

Remark: We propose, that when a user of ReMaP has access to data, that needs to be protected, they should be made aware / sign a document, that says that:

- They may not publish this data
- When they use this data to create simulations, they may not share these simulations with others

3.9.5.4 Ideas for Future

The following gives a list of possible new features for the future:

1. Start SFW with an Event in Venios
2. Run an instance of the SFW on the Venios platform (that can be started with 1.)
3. Communication between Control Algorithm and SFW over Venios (in this case the question is open, who would be the master of the process).
4. UI for running an experiment with different configurations for the components (with real HW, with simulation)

3.9.5.5 Tasklist SCS (Integration SFW)

Overview

This wiki page contains all features of the CFW (Control Framework), that are needed to integrate the Simulation Framework in ReMaP.

Legend:

✅ means: this feature/requirement can be implemented within the ReMaP budget

❌ means: it is not clear yet, if the feature/requirement can be implemented within the ReMaP budget (possibly not).

Connect SFW with real HW at Empa & PSI

SFW receives feedback, if the set control value has been set (at the machine) ✅

Requirement
When SFW sets control signal, it needs feedback, when the value has been set

Current Status
SCS provided a code example, how this check can be achieved: https://github.com/ReMaP-CH/remap-controller-example/commit/007968b974ceb15a57d0b5aa1cba6516f5889973

It is still open, if more is needed.

Reference
Direct communication between Diamantis and Sabine.

SFW can perform a HW-Connection-Check ✅

Requirement
Before starting a HiL simulation, there needs to be a check, if the used HW-signals can be

- read
- controlled

or generally, if the system is ready.

Current Status

Implemented. How it can be used, can be seen in the remap_service.py:

https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/remap_service.py

Reference
Protocol 2020-06-03: Integration Simulation Framework - Jun 2020[22] Point 6.

SFW can save its simulated Measurements in Venios ✅

Requirement
The SFW needs to be able to save timeseries data in Venios.

Current Status
Open.

Specification
SCS provides examples, on how the SFW can save simulated measurements in Venios.

Configure Signals in Python-Code ✅

Requirement
It should be possible to configure the read and control signals in a "comfortable" way.

Current Status
Implemented. As an example, one can look at the configuration for the ehub-Testcase:

https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/ehub/example_signals.py

Decoupling & Automization over Venios

SFW can execute Control Algo over Venios ❌

Requirement
The goal is, to be able to call a control algorithm (resp. its logic) from within the simulation framework, over Venios.

Thereby the control algorithm could run in any language, especially Python, C, Matlab, LabView, Java, C#.

Specification
The current idea for the implementation would include:

- define two jobs in Venios:
  - start control algo (SFW -> Control Algo)
  - send result of control algo (Control Algo -> SFW)
- Write wrapper code for control-algo, so that it can easily be integrated with Venios.
- Example code for SFW

In this scenario the Control Algo still needs to be started locally on the researchers machine. Being able to execute it in the cloud, could be a next step. Be aware though, that this would implicate that debugging (finding errors) in the control logic is more difficult, when it is run in the cloud directly. To make this really user-friendly would be a big effort.

Implementation / Tasks
- Specification/Clarifications:
  (Maybe the idea with the two jobs is not the best approach, this has to be discussed with Venios, the rest of the estimate is probably similar, since most work is providing examples in different programming languages)

22 https://tools.scs.ch/wiki/display/SGSREMAP/Protocol+2020-06-03%3A+Integration+Simulation+Framework+-+Jun+2020

- Implementation Venios
- Example Code SFW
- Example Code in several languages: Python, C, Matlab, LabView, Java, C#
  (budget depends on how many languages)

Reference
- Protocol 2020-06-03: Integration Simulation Framework - Jun 2020[23] point 3 "Access to external code".
- (SGS-intern): SGSRMSP-36[24]

User can start SFW over Venios ❌

Requirement
The idea is to start the SFW by sending a specific job (containing the input parameters for the SFW) to Venios.

Reference
Protocol 2020-06-03: Integration Simulation Framework - Jun 2020[25] point 5

SFW runs on Venios platform ❌

Requirement
Let the SFW run on the Venios platform in its own Docker Container. (This has **not** been stated yet as a requirement by the SFW-Team).

SFW can receive Control Signal from Venios ❌

Requirement
Similar to HW, the SFW could receive Control-Signals. (Unclear if this is actually needed).

User Interface

User Interface to start the SFW locally ❌

Requirement
Reference
Protocol 2020-06-03: Integration Simulation Framework - Jun 2020[26] point 5

3.9.6 Integration City Energy Analyst (CEA)

3.9.6.1 Current State of discussion

The following is a summary of what has been agreed on in a call on 12 Feb 2020 with:

- Sabine Proll[27]
- ETHZ - Shan-Shan Hsieh[28]
- SGS - Jimeno Fonseca[29]

23 https://tools.scs.ch/wiki/display/SGSREMAP/Protocol+2020-06-03%3A+Integration+Simulation+Framework+-+Jun+2020
24 https://tools.scs.ch/issues/browse/SGSRMSP-36
25 https://tools.scs.ch/wiki/display/SGSREMAP/Protocol+2020-06-03%3A+Integration+Simulation+Framework+-+Jun+2020
26 https://tools.scs.ch/wiki/display/SGSREMAP/Protocol+2020-06-03%3A+Integration+Simulation+Framework+-+Jun+2020
27 https://tools.scs.ch/wiki/display/~sproll
28 https://tools.scs.ch/wiki/display/~shsieh
29 https://tools.scs.ch/wiki/display/~JFonseca

3.9.6.2 Timeline CEA

The work for T3.4 is planned between June 2020-September 2021.

3.9.6.3 Minimal Integration

The simplest integration between the City Energy Analyst (CEA) and ReMaP is:

- Export of data from PSI and Empa (out of Venios) to be used in CEA → It needs to be checked, if the export data has any usage restrictions. If so, if not stated otherwise, the same restrictions as for the archive data, need to be applied to the resulting simulation.
- Import of (demand) data from CEA into Venios (TODO: ETHZ - Shan-Shan Hsieh[30] will provide an example of demand data to be stored in Venios)

SCS can provide the respective API-calls for export and import.

3.9.6.4 Howto for CEA integration

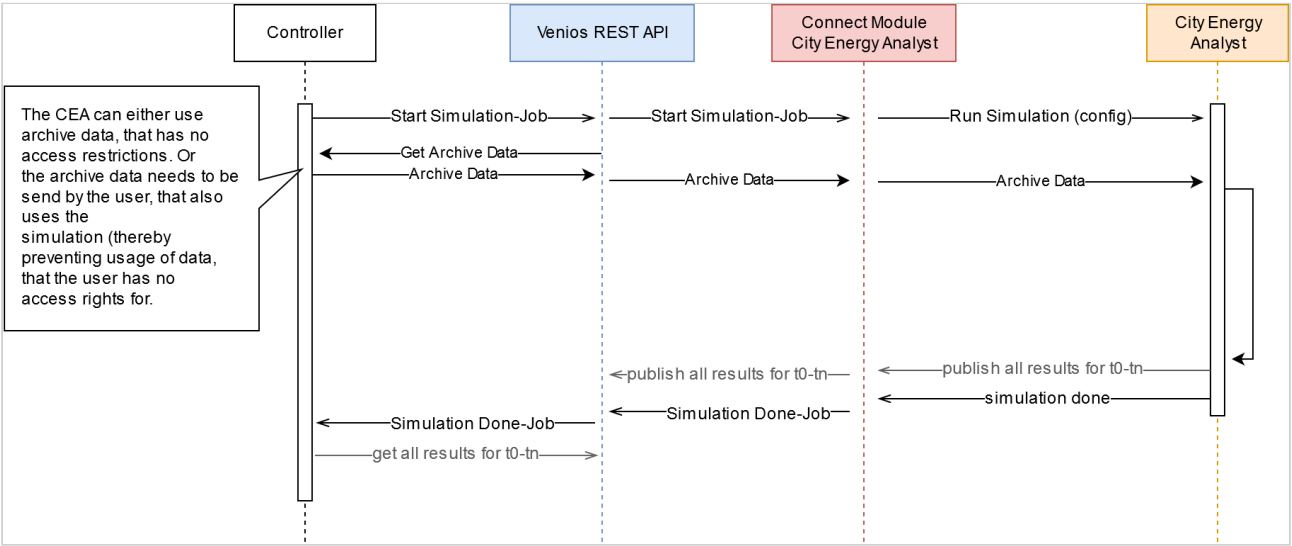
(Copy from Mail from Sabine to Shanshan)

- **Example Code** https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/CEA/test_example_calls_CEA.py
  - Save historic timeseries in Venios: see *test_log_measurements_with_timestamps()*
  - Retrieve these timeseries from Venios: see *test_read_measurements_from_venios()*
- **Setup Development** Setup Development[31]
  - documentation on how to checkout the code, run it under PyCharm, …
- **Access to Venios**
  - the first time you execute one of the tests it will crash and create a new file “access.ini”. This is where you need to add the credentials I sent you.
- **Define Signals**
  - General documentation Python Example Code[32]
  - You define your signals here: https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/CEA/example_signals_CEA.py
    as an example I added one signal, that can be logged (as log_signal) and read (as read_signal).
  - **venios_id**: You can choose your own venios_id for each signal. But please make sure it always starts with “CEA.” so that it is clear where the signal belongs to. Best practice is to have a hierarchy with “.” as separator. Here[33] you see a good example, how other systems have their names.
- **Possible Problem with multiple users**
  - adding measurements for a specific signal can only be done by the user who created the device. Reading can be done with all users assigned to the experiment.
- **Timeinterval for timeseries**
  - I set this to 2000-01-01 to 2030-01-01 for your experiment. If you have timestamps outside this interval let me know, I can easily adjust it.

30 https://tools.scs.ch/wiki/display/~shsieh
31 https://tools.scs.ch/wiki/display/SGSREMAP/Setup+Development
32 https://tools.scs.ch/wiki/display/SGSREMAP/Python+Example+Code
33 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/esi/ESI_3_10_signals.py

3.9.6.5 Optimal Integration

The following integration would be the optimum. If this will be done within the current ReMaP project still has to be decided. Ideally, one could use the same Connect Module as for the Simulation Framework (Integration Simulation Framework(see page 38)) and only change a small layer, interacting with the simulation.



3.9.6.6 Example_files_from_CEA

This folder includes a sample demand output from the CEA.

- Energy demand of each building is stored in B00x.csv
- A summary file called Total_demand.csv contains information of all buildings in the district.

T3.4 will deliver multiple scenarios for each typical district.

Please download this folder to see the current CEA folder structure of one district with four scenarios. You can find the demand data for each scenario here: “CEA_District_A/ScenarioX/outputs/demand” https://polybox.ethz.ch/index.php/s/pxzFdwE6fsylKAv

File	Modified
B01.csv[34]	Jul 28, 2021 by Sabine Proll[35]
Total_demand.csv[36]	Jul 28, 2021 by Sabine Proll[37]

34 https://tools.scs.ch/wiki/download/attachments/126527534/B01.csv?api=v2
35 https://tools.scs.ch/wiki/display/~sproll
36 https://tools.scs.ch/wiki/download/attachments/126527534/Total_demand.csv?api=v2
37 https://tools.scs.ch/wiki/display/~sproll

3.10 Regression Tests

3.10.1 Tests and Checks

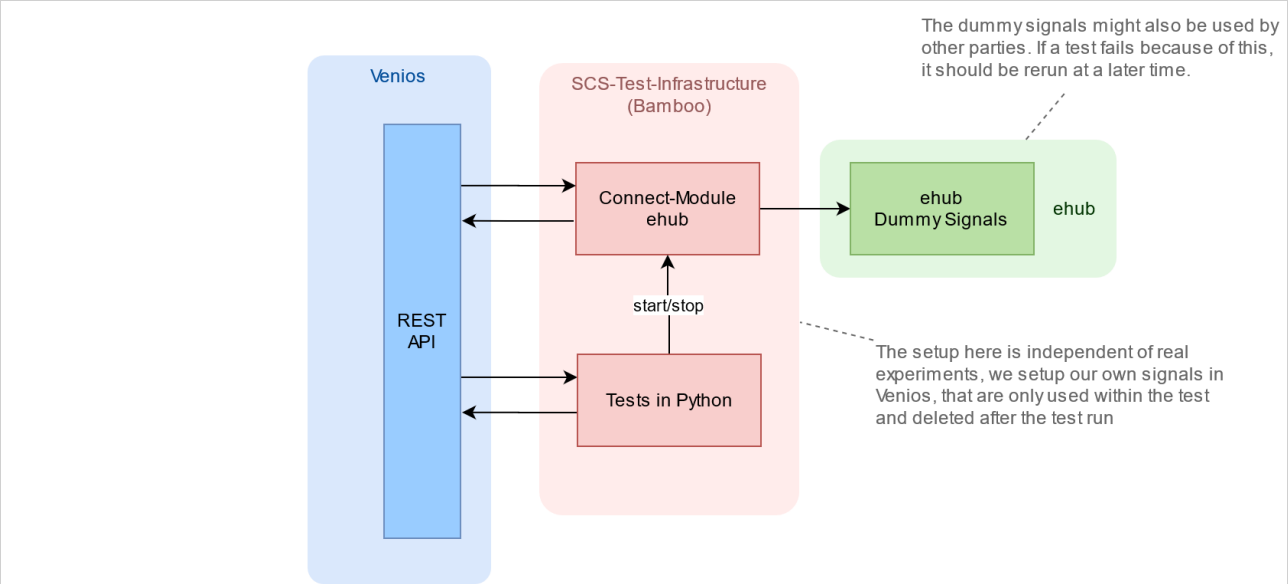
- Simple control-test for Dummy-Signals at ehub and ESI, which read one signal and control another signal, based on a simple control logic. Both tests runs for 40sec and execute the control logic every 5 sec.
  - ehub:
    - test: https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/ehub/test_run_example_controller.py
    - control logic: https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/ehub/example_controller.py
  - ESI:
    - test: https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/esi/test_run_example_controller.py
    - control logic: https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/esi/example_controller.py
- Checks at start of experiment (already used by the SFW).
  Code: https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/checks.py
  - Can read signals be read? → check_signals_being_archived ()
    Checks if all used read-signals return a value with a recent timestamp.
  - Is the system ready for me to control all control signals? → wait_for_control_access ()
    Waits until control-access is given by ehub/ESI for all control signals, or runs into a timeout-error.
- Unit tests for most of the functionality in the PythonExampleCode and the ConnectModules. The (unit)-tests of the ConnectModules and the PythonExampleCode run automatically with every new commit.

3.10.2 System Tests

The idea is, that SCS creates a setup for system tests, which can be used in the future to write more tests, especially regression tests. Which means, whenever a bug is discovered in any SCS-component (ConnectModule or PythonExampleCode) or Venios, it should be possible to reproduce it in a new system test.

The typical system tests will connect to dummy signals at Empa. The setup of ConnectModule and Python-Tests runs on a SCS-Buildserver (Bamboo). Basically the test in Python is the master, it starts the ConnectModule, then executes the actual test(s), and then stops the ConnectModule again.

The tests will use the test-system of Venios, and use its own signals, with their own VeniosIds. The tests will run on a nightly build.



SCS will start off with implementing a fix test-setup, with a fix configuration for the ConnectModule. The first set of system tests will cover:

1. Basic connection checks in Python:
    a. Can signals be read?
    b. Can control-signals be controlled?
    c. Can log-signals be logged from Python?
2. Checks on a Dummy-Experiment with a simple control logic.
   Throughout the experiment data is being collected. After the experiment the following points are checked:
    a. What was the max. time it took until a control-signal reached ehub? Is this time below a certain threashold (e. g. 1sec)?
    b. For all actions (read, control, log): was the latency of each call to Venios below a certain threashold (e. g. 300ms)?

3.10.3 Brainstorming / Ideas (deprecated)

Possible Problems:

- HW broken
    - Control
    - Read
- Communication broken / slow
- System Configuration in SFW
- Control Algorithm broken/wrong
- SFW: Models are incorrect (don't work as expected)

Workflow / Debugging from User-Perspective

- Configuration
- Performance

Testing Idea	**Problems, that are discovered**	**Next steps**
System Tests • Generic tests written in SFW, connected to Dummy Signals • Incl. checking of latency in the system *Details (Mail Sabine):* *I think we should implement a fake experiment with the SFW, which controls only Dummy-Signals at ehub and ESI. This way we test the complete system, except the real hardware.* *This fake experiment should be similar to real experiments w. r. t.:* • *Amount of data* • *Frequency of calculation (How many steps are performed in the SFW per minute?)* • *Interaction with Venios (logging of simulated data, controlling hardware, …)* • *Duration of Experiment* *We could also make this configurable, depending on the needs of the next experiment(s). E. g. if a next experiment needs fast latency, one could configure the test, such that Steps per Minute is as high, as it is needed for the experiment.*	• Broken signal transfer (between SFW - Venios - ConnectModule - ehub/ ESI) • Bad Latency for signal tranfer (Anywhere between SFW - Venios - ConnectModule - ehub/ ESI)	

Testing for the Experiment Code
• Integration of single HW in SFW (no connection with other components)
  • Testing of HW-Model in SFW (connected to real HW)
• SFW without HW
  • Simulation Model (inside SFW) of HW for Testing (Testing without Control Framework)
  • Connected to Dummy Signals
	• Connection to specific HW
	• Problems in Experiment Code in combination with SFW
	• Problems anywhere between SFW - Venios - ConnectModule - ehub/ ESI	

System Check before Experiment Start This is something we already have, and I'm not sure if we need more here. At the start of the experiment we do check, if all signals can be read and if the needed control-access is granted.	• Connection to all HW-components, generally ok (write access is given, read signals are current) • Configuration of signals is ok	nothing planned

Testing Idea	Problems, that are discovered	Next steps
Test Suites for all ReMaP-Software Components All software components (SFW, ConnectModules, Components in Venios, …) should have automated tests. Currently nobody is coordinating this / making sure, that this happens. I think Andreas (or rather his successor) could make sure, that this is really done.		nothing planned

4 Erfüllung Requirements

4.1 Abnahme Protokoll Juni 2021

Am 30 Jun 2021 hat eine Abnahme zu den Arbeiten von SGS (mit Venios und SCS) für ReMaP Phase I stattgefunden. Das folgende Dokument wird von allen Parteien, welche teilgenommen haben, unterschrieben:

Hier ist insbesondere der finale Stand und die offenen Mängel festgehalten.

4.2 ReMaP Phase I - komplett

Im folgenden die Requirements welche, nach aktuellem Plan, in ReMaP Phase I umgesetzt werden können.

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Functional Requirements			
	Control of Hardware			

WP		Originally planned	Implemented (06-2021)	Acceptance by
T2.1/.2	Control at ehub	✅ • Connection to ehub • 1 type of hardware • 1-5 control values per hardware	✅	Proven by running experiments T3.5 and T3.10
	Control at ESI	✅ • Connection to ESI • 1 type of hardware from ESI • 1-5 control values per hardware	✅	Proven by running experiment T3.10
	Control at ehub and PSI at the same time	✅	✅	Proven by running experiment T3.10 (Control of battery at ehub and Electrolyzer at ESI)
	Archive			

WP		Originally planned	Implemented (06-2021)	Acceptance by
T2.1/.2	Persisting Data Streams	✅ • Data from ehub • 1 type of hardware of ehub • 5-10 sensor values for the hardware • Data PSI • 1 type of hardware of ESI • 5-10 sensor values for the hardware	✅	▪ ehub: ▪ 13 Oct 2020 Philippe Buchecker confirms, that 80 signals from the CHP of T3.5 were recorded continuously over the period of 2 weeks: Re_ Abnahme ehub-Anbindung.msg[38] ▪ ESI: ▪ 03 Sep 2020 connection of ESI Dummy signals shown in video by Markus Obrist: ReMaP dummy connection to ESI.msg[39] ▪ 30 Oct 2020 connection of all ESI signals with Venios (faked values on PSI-side, but real connection between ESI and Venios), confirmed by Markus Obrist: Re_ Abnahme_ Anbindung ESI - Venios Teil 2.msg[40] ▪ Also proven by running experiments T3.5 and T3.10
	Persisting Bulk Data	❌	-	
	Accessing the archive	✅	✅	Proven by running experiment T3.10

38 https://tools.scs.ch/wiki/download/attachments/74716846/Re_%20Abnahme%20ehub-Anbindung.msg?api=v2&modificationDate=1624550932722&version=1
39 https://tools.scs.ch/wiki/download/attachments/74716846/ReMaP%20dummy%20connection%20to%20ESI.msg?api=v2&modificationDate=1624551486397&version=1
40 https://tools.scs.ch/wiki/download/attachments/74716846/Re_%20Abnahme_%20Anbindung%20ESI%20-%20Venios%20Teil%202.msg?api=v2&modificationDate=1624552344016&version=2

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Deleting data in the archive	✅	✅ Remark: Can be done by SCS, by deleting either single measurements, or deleting a MeasurementDevice (whereas the MeasurementDevice itself has to be deleted + any references to it from any Experiment).	Can be executed by SCS. SCS confirms that the feature is implemented.
	Annotations for Experiments	❌	💡it is possible to check if a measurement or device belongs to an experiment.	
	Export of ehub-Archive to ReMaP	❌	-	
	Backup	❌ Currently not planned, but should be evaluated later again. Probably not a big effort to do this.	-	
	Simulation			

WP		Originally planned	Implemented (06-2021)	Acceptance by
T2.1/.2	Integration of Simulation Framework	✅	💡 the requirements from Tasklist SCS (Integration SFW) - Aug 2020[41] were implemented (see below "Simulation - "Tasklist SCS" (as fixed 07-201)	
	Simulation and Usage of access-restricted Archive Data	✅	✅	SCS confirms that the feature is implemented.
	Testing Control Algorithm against Simulation	✅	✅ this requirement is fullfilled by FEN with the implementation of the simulation framework, not the control framework.	
	Scaled Mocks of real Hardware	**IN PROGRESS**	Generally possible, but nothing was implemented to make this more convenient.	

41 https://tools.scs.ch/wiki/display/SGSREMAP/Tasklist+SCS+%28Integration+SFW%29+-+Aug+2020

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Simulation - "Tasklist SCS" (as fixed 07-2021)	Tasklist SCS (Integration SFW) - Aug 2020[42] Requirements were approved by Diamantis Marinakis per e-mail on 02 Jul 2020		
T2.1/.2	SFW receives feedback, if the set control value has been set (at the machine)	✅	✅	• 29 Jun 2021 Diamantis Marinakis confirms implementation of all 4 features per e-mail: RE Acceptance for the integration of the SFW.msg[43] • Also proven by running experiments T3.5 and T3.10
	SFW can perform a HW-Connection-Check	✅	✅	
	SFW can save its simulated Measurements in Venios	✅	✅	

42 https://tools.scs.ch/wiki/display/SGSREMAP/Tasklist+SCS+%28Integration+SFW%29+-+Aug+2020
43https://tools.scs.ch/wiki/download/attachments/74716846/RE%20Acceptance%20for%20the%20integration%20of%20the%20SFW.msg?api=v2&modificationDate=1625037624652&version=1

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Configure Signals in Python-Code	✅	✅	
	SFW can execute Control Algo over Venios	❌	-	
	User can start SFW over Venios	❌	-	
	SFW runs on Venios platform	❌	-	
	SFW can receive Control Signal from Venios	❌	-	

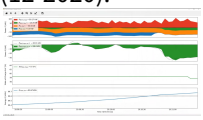
WP		Originally planned	Implemented (06-2021)	Acceptance by
	User Interface to start the SFW locally	✗	-	
	Rights Management			
T2.1/.2	Login	✓	✓	FEN, ETH, Empa, PSI Shown by experiment T3.5 and T3.10
	Synchronisation Users ehub ↔ ReMaP	✗	-	
	Execution of an Experiment	✓	✓	FEN, PSI Shown by last experiment T3.10

WP		Originally planned	Implemented (06-2021)	Acceptance by
	No write access to Actors outside an Experiment	✅	✅	Can be executed by SCS. SCS confirms that the features are implemented. Furthermore SCS will provide (automated) test cases. Review can be done by anyone, who has used the Python Codebase.
	Read Access to Sensors outside an Experiment	✅	✅ can be accomplished by defining a long running "experiment" with the corresponding devices	
	Full Access to Measurements of an Experiment	✅	✅	
	Restricted Access to Measurements of an Experiment	✅	✅	

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Grant read-access to an Experiment	✅	✅	
	Combination read access and read and write access per experiment	✅	✅	
	Full Read-Access to Subset of Measurements of an Experiment	❌	✅	
	User Registration	✅	✅ has to be done manually by SCS and Venios	SCS confirms that the features are implemented.
	Grant Access Right - Only Venios	✅	✅	SCS confirms that the features are implemented.

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Grant Access Right - ReMaP Admin	✅	✅	SCS confirms that the features are implemented.
	Several Experiments running on different Hardware Resources at same time	✅	✅	SCS confirms that the features are implemented. TBD, can be done but so far no 2 experiments were running at the same time.
	Manual Prevention of simultaneous Control of same Hardware Resource	✅	✅	SCS confirms that the features are implemented.

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Complete Prevention of Simultaneous Experiments running on same Hardware Resource	✗	-	
	Read-Access for non-ReMaP-Admins to current and future hardware control	✗	✓	All users can execute the script get_experiments https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/scripts/get_experiments.py and get the information on all experiments with their corresponding assigned users, devices and time interval. (Open: add actual name of the user to the UserModel-object in field "name" of UserModel (currently this field is filled with the ID - the matching between ID and actual person is currently done in a table in the Wiki maintained by SCS and Venios). It will be documented how to do this, so that Empa and PSI know how they can check the set rights.
	Scheduling	✗	-	

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Improvements for User Experience			
T2.4	Overview	❌	💡 Generally most of the information is given somewhere, but not as a unified overview	
T2.4	Viewer	✅	💡 It was decided in January 2021, that no viewer will be implemented. There is Python code from SFW, Empa and PSI, that shows how current measurements can be displayed: ▪ SFW displays current measurements on the command line ▪ Empa implemented a desktop app for the demo case (12-2020): 	

WP		Originally planned	Implemented (06-2021)	Acceptance by
T2.1/.2	Assistance for Coding	(✅) Basic assistance: ▪ SoapUI template for 2 usecases ▪ Example code for 2 usecases	💡 instead checks were implemented to check the connection to hardware (see above "Simulation - "Tasklist SCS" (as fixed 07-2021)")	

WP		Originally planned	Implemented (06-2021)	Acceptance by

	System monitoring	(✅)
		For Administrators
		• Logs of the ReMaP Software ✅
		• network connection ✅
		• Latency in network ❌
		• Health state of all ReMaP software components ✅ (gegeben durch Venios)
		• Connection state:
		  • between Connect Module and ehub/ESI ✅
		For Researchers:
		• Access to
		  • received measurements ✅
		  • sent control values ✅
		• Logs from the DUT (device under test, i. e. "the hardware") ❌
			✅
			For Administrators:
			▪ All information is given in the log-files of the ConnectModules, specifically:
			  ▪ Connection to Venios
			  ▪ Connection to ehub/ESI
			  ▪ Thereby also giving the health state of all 3 components (Connect Module, Venios, ehub/ESI)
			For Researchers:
			▪ When using the Python Codebase the user gets information on:
			  ▪ connection to read signals
			  ▪ connection to write signals
			(see also above "Simulation - "Tasklist SCS" (as fixed 07-2021)")
				For Researchers:
				(see above "Simulation - "Tasklist SCS" (as fixed 07-2021)")

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Non-Functional Requirements			
	System Performance			
T2.1/.2	Latency of 1 sec	(✅) will not be reached at all devices at PSI	💡 currently signals are sampled at a 2 sec sampling rate at ehub and at 5 sec at ESI. Resulting in a Latency of approx. Samplingrate + max. 1 sec. Signals are sampled "on change", so in case of "rare" changes the latency of 1 sec can actually be achieved. Maybe a better configuration on what "a change" is, would result in better results.	▪ 30 Jun 2021 Max Wurm (Venios) confirms that they can commit to a sampling of ▪ 1000 signals ▪ with samplingrate 1sec ▪ and <5% rejects (measurements that cannot be sampled) RE Latenz Venios.msg[44] Observations on the current latency will be delivered later, as part of the regression tests. To show the actual latency of the system as a whole, we would need to collect data from ehub/ESI and a running experiment (possibly from the simulation framework) and compare the timestamps.
	Latency < 1 sec	❌	-	

44 https://tools.scs.ch/wiki/download/attachments/74716846/RE%20Latenz%20Venios.msg?api=v2&modificationDate=1625045314868&version=1

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Timestamp precision 1sec	✅	✅	To show the actual timestamp precision, the timestamp in Venios would have to be compared with timestamps directly collected at ehub/ESI.
	Timestamp precision < 1sec	❌	The actual time precision is probably at around 50-200ms. But this has not been proven.	
	Ease of extensibility			
T2.1/.2	Add new hardware	✅	✅ New hardware can be added by adding it to the signallist and generating a new config for the ConnectModule. See Integration of External Systems - Mar 2021[45]	▪ 30 Oct 2020 Confirmation by Markus Obrist (PSI): Re_ Abnahme_ Anbindung ESI - Venios Teil 2.msg[46] ▪ 14 Sep 2020 DeviceConfig for ConnectModule could be generated by Empa, see mail by Sascha Stoller: Signale Empa.msg[47]
	Add new simulations	✅	✅ Provided by the simulation framework	

45 https://tools.scs.ch/wiki/display/SGSREMAP/Integration+of+External+Systems+-+Mar+2021
46 https://tools.scs.ch/wiki/download/attachments/74716846/Re_%20Abnahme_%20Anbindung%20ESI%20-%20Venios%20Teil%202.msg?api=v2&modificationDate=1624552344016&version=2
47 https://tools.scs.ch/wiki/download/attachments/74716846/Signale%20Empa.msg?api=v2&modificationDate=1624552834146&version=1

WP		Originally planned	Implemented (06-2021)	Acceptance by
	Add new experiments	✅	✅ New experiments can be added by SCS. To create a new experiment, the following has to be provided: ▪ signal list (as used in the Python code) ▪ time interval ▪ list of users	PSI (Markus Obrist)
	Security	**IN PROGRESS**	✅ see IT Agreement between PSI, SGS (Venios, SCS)	PSI (Marcel Hofer, Markus Obrist), SGS (Roland Lüthy)
	Testing	see Regression Tests(see page 45)		
	Tests and Checks	✅	💡 will be implemented. Currently still open is to automatically run the tests of the Python Code on a build server	
	System Tests	✅	💡 will be implemented.	
	Other Work Packages			

WP		Originally planned	Implemented (06-2021)	Acceptance by
T1.4	Measurement System (incl. Smart Boxes)		• Smartboxes at Empa: Implementation of the smartboxes from the backend to the Venios platform is complete, the boxes have been delivered 17.06.21	Empa Silvestre Scheider, the installation of the boxes, which was carried out by Empa, is still open all the smartboxes are tested incl communikation

5 Glossar

5.1 Energy sector

Term	Meaning
RES	Renewable Energy System
SCCER FEEB&D	Swiss Competence Center for Energy Research on Future Energy Efficient Buildings & Districts http://www.sccer-feebd.ch/
CEDA	Coherent Energy Demonstrator Assessment https://www.sccer-mobility.ch/Joint_Activities/CEDA-Coherent-Energy-Demonstrator-Assessment/

5.2 Technical

Term	Meaning	Synonym / Translation
PLC	Programmable Logic Controller	SPS = Speicherprogrammierbare Steuerung

5.3 Project-specific

Term	Meaning
CFW	Control FrameWork
SFW	Simulation FrameWork

7.2 Operation manual for Control Framework

See File: SCS-Wiki-ReMaP -Operations-v3-20230224_152008.pdf

Operations

SGS REMAP

Exported on 02/24/2023

Table of Contents

1 General

1.1 Basic Venios Access via Browser

The easiest way to access Venios is via a Browser: https://remap.venios.de:8282

Here you can execute all calls to Venios by unfolding a command and clicking "Try it out".

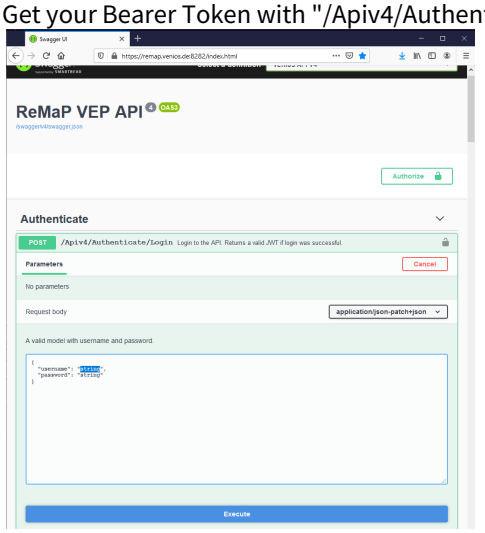
Important calls:

- The first step is to Authenticate(see page 6)
- To access a device, you need to Get the UUID(see page 8)

1.1.1 Authenticate

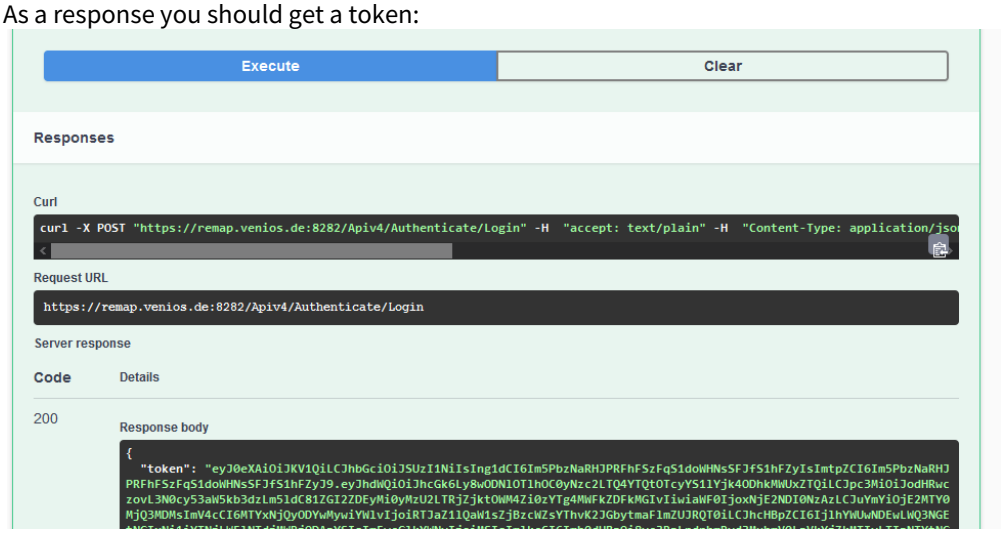
Before being able to execute any calls, you first have to get a login token:

1. Get your Bearer Token with "/Apiv4/Authenticate/Login"

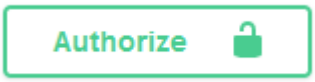


   Type in your username and password in the white window and click "execute"

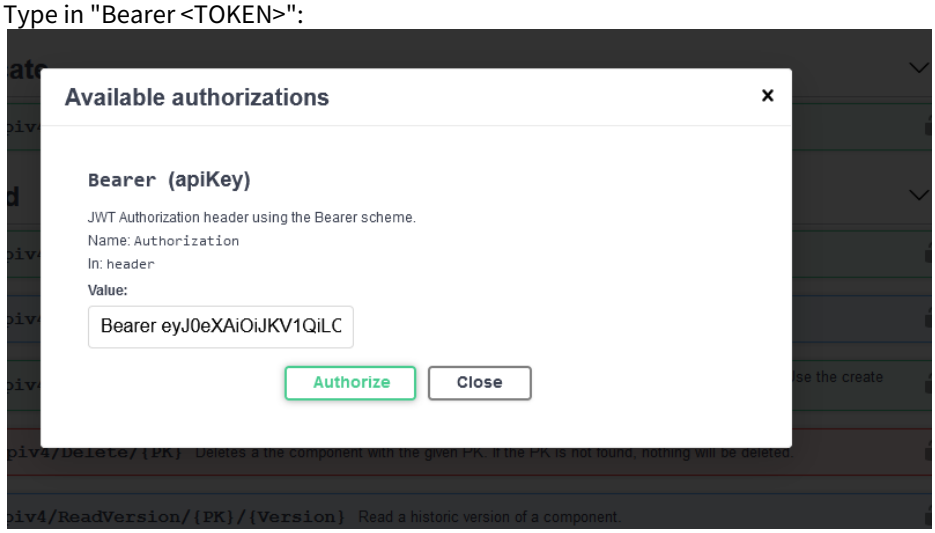
2. As a response you should get a token:



3. Go to the top of the page and click "Authorize"

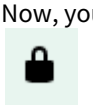


4. Type in "Bearer <TOKEN>":



and click Authorize

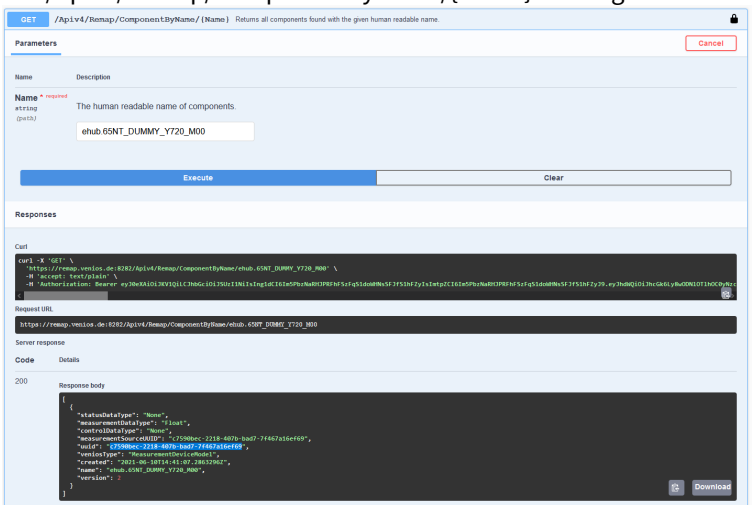
5. Now, you should be able to execute all calls (check if the locks are all closed:



this means you are logged in.

1.1.2 Get UUID

- Call /Apiv4/Remap/ComponentByName/{Name} with a given VeniosId

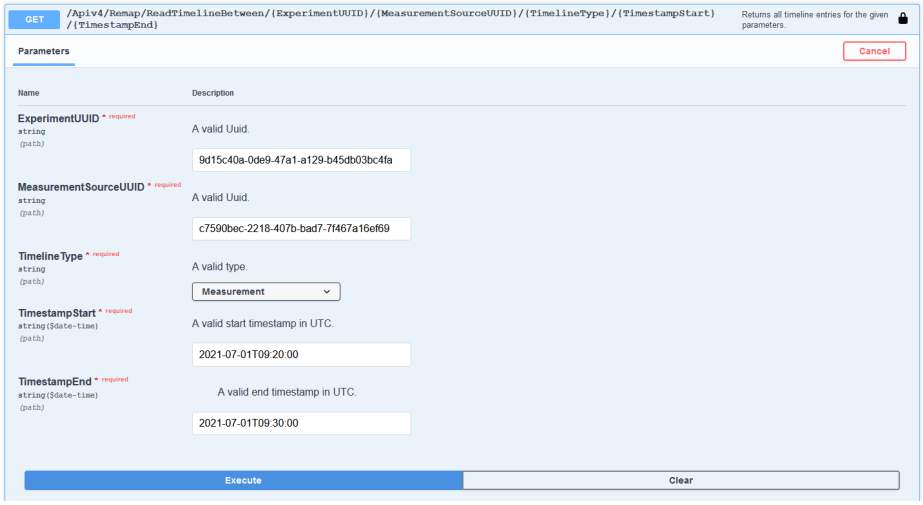


Copy the uuid

1.1.3 Get Timelines

1.1.3.1 For Time Interval

- Open call /Apiv4/Remap/ReadTimelineBetween/{ExperimentUUID}/{MeasurementSourceUUID}/{TimelineType}/{TimestampStart}/{TimestampEnd}



and fill out:
ExperimentUUID: Must be provided to you by the ReMaP Admin.
If the access is setup for you, you can also get the experiment_uuid from here: current list of experiments,[1]
also check Current Access Rights(see page 11)
MeasurementSourceUUID: uuid from call above

1 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/scripts/current_experiments.json

TimelineType: typically Measurement (for float) or Status (for int and bool)
TimestampStart/End: see image for format. Beware that timestamps in Venios are always in UTC.

1.1.3.2 Most Current Value

- Open call /Apiv4/Remap/ReadTimelineLatest/{ExperimentUUID}/{MeasurementSourceUUID}/{TimelineType}
  and fill out:
  ExperimentUUID: Must be provided to you by the ReMaP Admin.
  If the access is setup for you, you can also get the experiment_uuid from here: current list of experiments,[2]
  also check Current Access Rights(see page 11)
  MeasurementSourceUUID: uuid from call above
  TimelineType: typically Measurement (for float) or Status (for int and bool)

1.1.4 Send Control Values

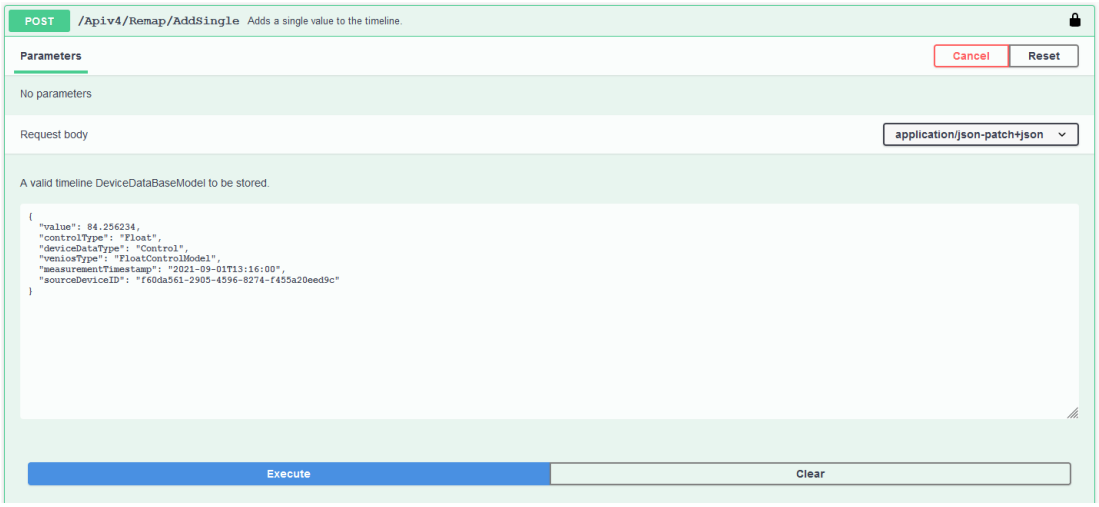
1.1.4.1 Send a Control Value

- Get the UUID by using Get UUID(see page 8) with the VeniosId of the ControlSignal (e. g. "ehub.65NT_DUMMY_Y720_Y00" or "ESI.DUMMY.Setpoint_Analog")
- To send the control value, means to do the following call:
  /Apiv4/Remap/AddSingle
  with body:

```
{
  "value": [control value],
  "controlType": "[type of value]",
  "deviceDataType": "Control",
  "veniosType": "[venios class name]",
  "measurementTimestamp": "[current timestamp in UTC]",
  "sourceDeviceID": "[UUID from ControlSignal]"
}
```

  the above can be copied, with the following fields adjusted:
  - **value** the control value to be set
  - **controlType** Float, Bool or Int (corresponding to veniosType)
  - **veniosType** FloatControlModel, BoolControlModel or IntControlModel (corresponding to controlType)
  - **measurementTimestamp** set to the current time in UTC
  - **sourceDeviceID** UUID from the ControlSignal

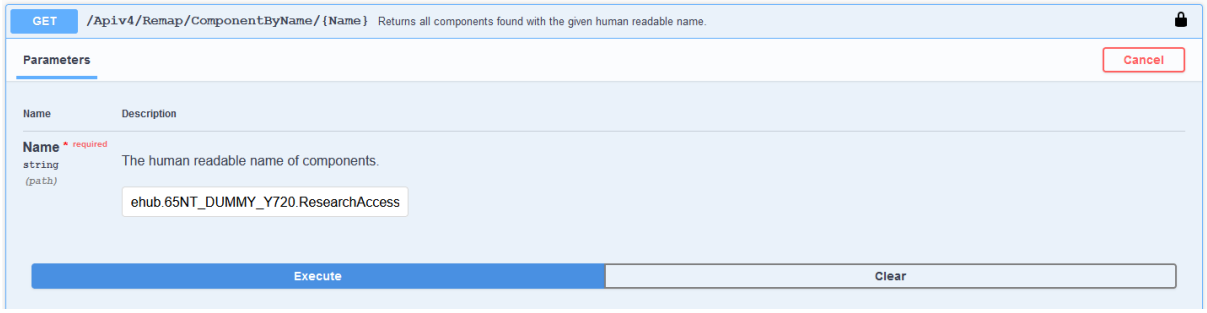
2 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/scripts/current_experiments.json



1.1.4.2 Continuously send research_mode_on

The control value is not send to the actual hardware, unless the "research_mode_on" is set continuously. Every 10 sec is sufficient.

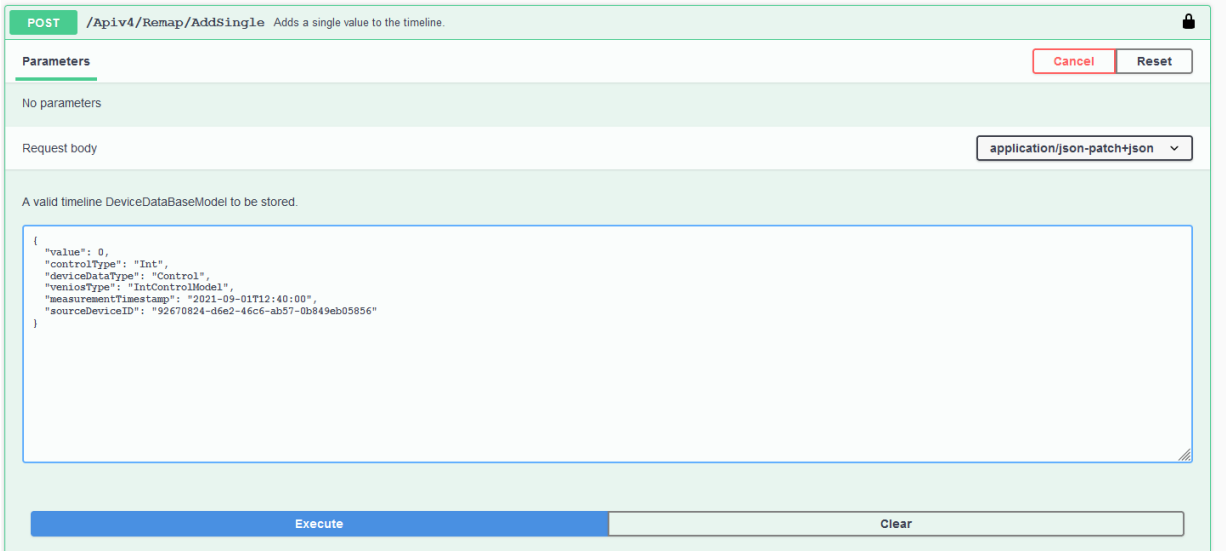
- Get the UUID by using Get UUID(see page 8) with the "access_id" (e. g. "ehub.65NT_DUMMY_Y720.ResearchAccess" or "ESI.DUMMY")



- To send "research_mode_on" means to do the following call:
  /Apiv4/Remap/AddSingle
  with body:

```
{
  "value": 0,
  "controlType": "Int",
  "deviceDataType": "Control",
  "veniosType": "IntControlModel",
  "measurementTimestamp": "[CurrentTimestamp in UTC]",
  "sourceDeviceID": "[UUID from AccessId]"
}
```

  the above can be copied, with the following fields adjusted:
  - measurementTimestamp: set to the current time in UTC
  - sourceDeviceID: set to UUID from the AccessId



1.2 Current Access Rights

1.2.1 Get rights information from github

The easiest way to check the current access rights, is to go to github https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/scripts/current_experiments.json

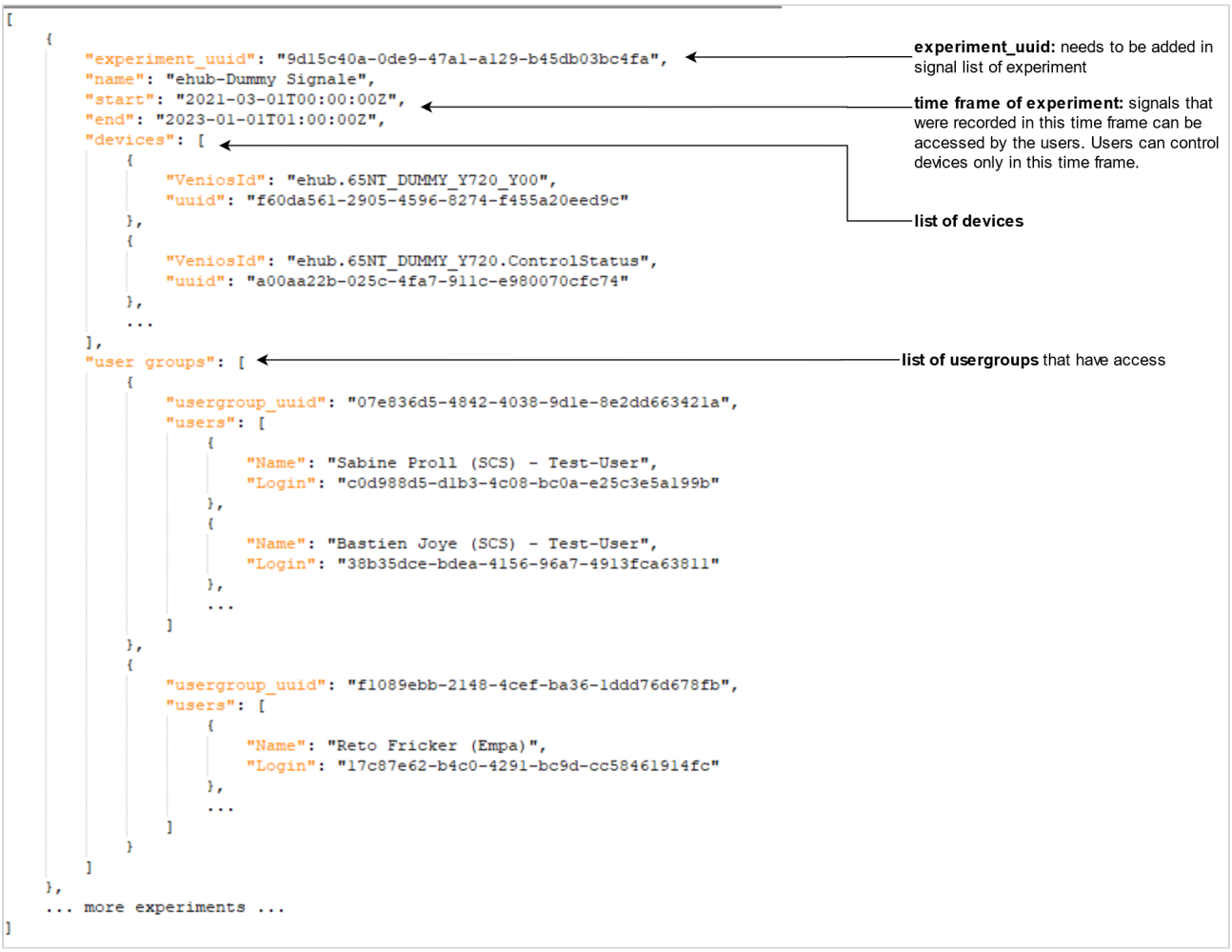
Each experiment has a time frame, devices assigned to it and a list of users, who can access the devices in the time frame.

This file will be kept up-to-date, whenever rights change.

1.2.2 Access rights information directly from Venios

To actually access this data directly from Venios, you have to install the Researcher Client (Python Example Code), as explained here: Setup Development(see page 27) and execute the following script https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/scripts/current_experiments.json

1.2.3 How to read the file



1.3 Github - Names

Mapping real name to github name

Real Name		Github Name
Maximilian Wurm	Venios	BaluJr
Markus Obrist	PSI	mobrist
Ruedi Limacher	SCS	rtrnrd
Sabine Proll	SCS	MaSaPr
Christian Schürch	ETH	schuercc

Real Name		Github Name
Bastian Joye	SCS	TheFlux7
Daren Thomas	ETH	daren-thomas
Diamantis Marinakis	ETH	Diamantis-FEN
Gianfranco Guidati	ETH	gguidati
Benjamin Huber	Empa	hubebenj
Philippe Buchecker	ETH	philippe-bu
Mario Coray	SCS	raymar9
Shanshan Hsieh	ETH	shanshanhsieh
Sascha Stoller	Empa	sstoller
Thierry Zufferey	ETH	thierryz88

2 Researchers

2.1 Python Example Code

2.1.1 Introduction

The Python Example Code[3] provides

- code, that simplifies the access to the Venios REST API
  https://github.com/ReMaP-CH/remap-controller-example/tree/master/controller/common
- examples, showing how to connect to Venios:
  https://github.com/ReMaP-CH/remap-controller-example/tree/master/controller/example_controller

2.1.1.1 Setup Development

See Setup Development(see page 27), for git checkout, how to use PyCharm, ...

2.1.2 Defining Signals

In the Python-code base we differentiate between 3 types of Signals:

- ReadSignal[4]
  A signal that is read from within the Python-Code
- ControlSignal[5]
  A signal that is controlled from within the Python-Code
- LogSignal[6]
  A signal that is logged from within the Python-Code. Used, when the researcher wants to save his own, simulated, measurements.
  (Of course, this could also be used to log measurements of a real hardware device in Venios).

3 https://github.com/ReMaP-CH/remap-controller-example
4 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/read_signal.py
5 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/control_signal.py
6 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/log_signal.py

To define your own signals.py, copy an exisiting list (for example: example_signals.py)[7], remove all the signals and update the experiment_uuid (see also Access Restriction[8]). An empty signals.py-file should look like this:

```
from common.control_signal import ControlSignal
from common.log_signal import LogSignal
from common.read_signal import ReadSignal

experiment_uuid = ''


all_read_signals = [read_signal for read_signal in locals().values() if
                    isinstance(read_signal, ReadSignal)]

all_control_status_signals = \
    [status_signal for status_signal in all_read_signals if
     (status_signal.classname_venios == "EHubStatusModel" and
      status_signal.fieldname_venios in {"researchModeConfirmation",
"researchModeApproval"}) or
     (status_signal.classname_venios == "PSIStatusModel" and
      status_signal.fieldname_venios == "controlApproved")]

all_control_signals = [control_signal for control_signal in locals().values() if
                       isinstance(control_signal, ControlSignal)]

all_log_signals = [log_signal for log_signal in locals().values() if
                       isinstance(log_signal, LogSignal)]
```

The variables on the bottom will contain all the signals of a specific type and are used for initialization (see examples below).

2.1.2.1 General

All signals have the following fields, that need to be filled:

- **veniosId** identifyer of the signal within ReMaP. Also gives a hierarchy on where the signal is located.
- **classname_venios, fieldname_venios, field_type** how the signal is saved in Venios. Always one of the following:

7 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/ehub/example_signals.py
8 https://tools.scs.ch/wiki/display/SGSREMAP/Integration+of+External+Systems#IntegrationofExternalSystems-AccessRestriction

classname_venios	fieldname_venios	field_type	Can be used as	
BoolDataModel FloatDataModel IntDataModel	value	bool float int	ReadSignal, LogSignal	
BoolStatusModel FloatStatusModel IntStatusModel	value	bool float int	ReadSignal, LogSignal	

classname_venios	fieldname_venios	field_type	Can be used as	
EHubStatusModel	alarmActive researchModeConfirmation researchModeApproval alarmStatus researchModeStatus	bool bool bool int int	ReadSignal	Status of Control-Device at Ehub
PSIStatusModel	realDataApproved cedaDataApproved controlApproved	bool bool bool	ReadSignal	Status of Control-Device at ESI
SignalStatusModel	error	bool	ReadSignal	Error data (only used at PSI)

classname_venios	fieldname_venios	field_type	Can be used as	
BoolControlModel FloatControlModel IntControlModel	value	bool float int	ControlSignal	Signal represents a control-value, that can be set over the Venios-REST-API.

2.1.2.2 Read Signal

Definition

The definition of a read signal is straightforward and can be derived from the ehub/ESI-Signallist:

```
read_signal_dummy_M00 = ReadSignal(
    venios_id="ehub.65NT_DUMMY_Y720_M00",
    classname_venios="FloatDataModel",
    fieldname_venios="value",
    field_type=float
)
```

Usage

```
current_value = venios_api.read(read_signal_dummy_M00)
```

2.1.2.3 Control Signal

Definition

A control signal has additionally

- access_id: used by ControlAccess[9] to send the research_mode_on, see here(see page 22)
- corresponding_read_signal: used by initialize_control[10] to initialize the control signal, see here(see page 21)

```
control_signal_dummy_Y00 = ControlSignal(
    venios_id="ehub.65NT_DUMMY_Y720_Y00",
    classname_venios="FloatControlModel",
    fieldname_venios="value",
    field_type=float,
    access_id="ehub.65NT_DUMMY_Y720.ResearchAccess",
    corresponding_read_signal=read_signal_dummy_M00
)
```

Also note, to be able to check, if the control access is successfull, certain ReadSignals need to be defined:

Example Ehub:

```
status_signal_dummy_ResearchModeConfirmation = ReadSignal(
    venios_id="ehub.65NT_DUMMY_Y720.ControlStatus",
    classname_venios="EHubStatusModel",
    fieldname_venios="researchModeConfirmation",
    field_type=bool
)
status_signal_dummy_ResearchModeApproval = ReadSignal(
    venios_id="ehub.65NT_DUMMY_Y720.ControlStatus",
    classname_venios="EHubStatusModel",
    fieldname_venios="researchModeApproval",
    field_type=bool
)
```

Example ESI:

```
status_signal_dummy_ControlApproved = ReadSignal(
    venios_id="ESI.DUMMY.Status",
    classname_venios="PSIStatusModel",
    fieldname_venios="controlApproved",
    field_type=bool
)
```

9 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/control_access.py
10 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/initialize_control.py

Usage

```
venios_api.control(control_signal=control_signal_dummy_Y00, value=26.3462)
```

2.1.2.4 Log Signal

Definition

A log signal does not contain additional fields. If the device does not exist yet, it has to be created, see here.(see page 14)

```
log_signal_dummy = LogSignal(
    venios_id="ehub.65NT_DUMMY_Y720_simulation",
    classname_venios="FloatDataModel",
    fieldname_venios="value",
    field_type=float,
)
```

Usage

```
venios_api.log(log_signal=log_signal_dummy, value=5)
```

2.1.3 Initialize Connection with Signals

The easiest way to access hardware from Python, is to use code provided by the simulation framework. If the simulation framework is not used, the best entry point to see how to initialize everything is the ehub-example-controller[11]

or from there the remap_service.py[12]. In RemapService.run() you see all the steps, that are needed to initialize the connection to hardware over Venios. We explain them here in detail:

2.1.3.1 Create (not-existing) Devices for Log Signals

For **log-signals** the corresponding devices need to be created, if not existing yet:

```
setup_log_measurement_devices(self.venios_api, all_log_signals, experiment_uuid)
```

11 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/example_controller/ehub/test_run_example_controller.py
12 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/remap_service.py

2.1.3.2 Initialize all Signals

To access **any signals** in Venios, we need to know its UUID and the experiment_uuid of the experiment. Both need to be set in our signal-objects, to be able to use them with the VeniosApi. This can be done with the function set_all_signals_uuids():

```
set_all_signals_uuids(self.venios_api, all_read_signals, all_control_status_signals,
all_control_signals,
                              all_log_signals, experiment_uuid)
```

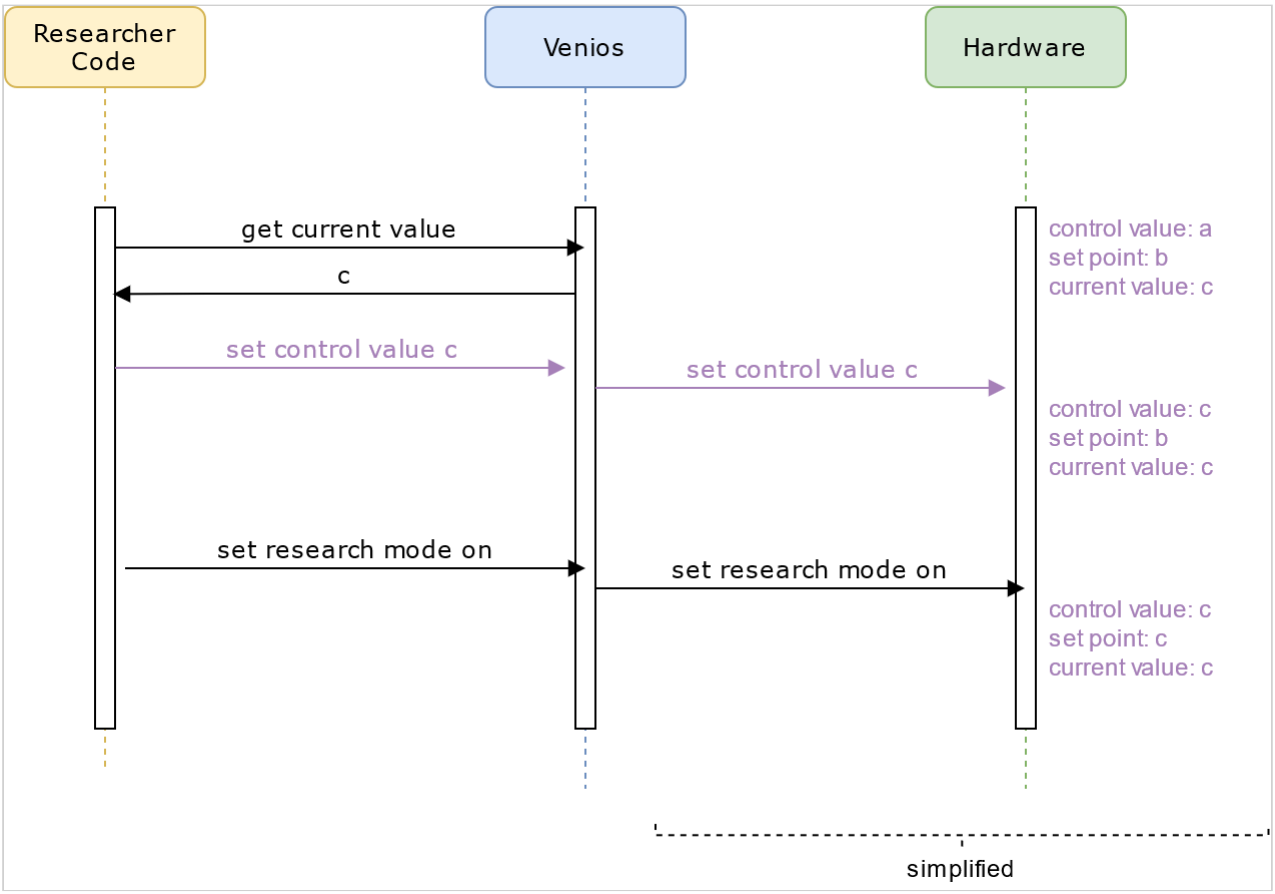
2.1.3.3 Check Read Signals for current Measurements

For all **read signals**, we **check** if there are current measurements. If not, it can be assumed that the signal is not properly connected and the code stops.

```
ready = True

# check that the connect module is running and archiving
ready &= check_signals_being_archived(self.venios_api, all_read_signals,
all_control_signals)

if not ready:
   print("System not ready for experiment: signals are not being archived")
   exit(1)
```

2.1.3.4 Control Access

Initialize with current value

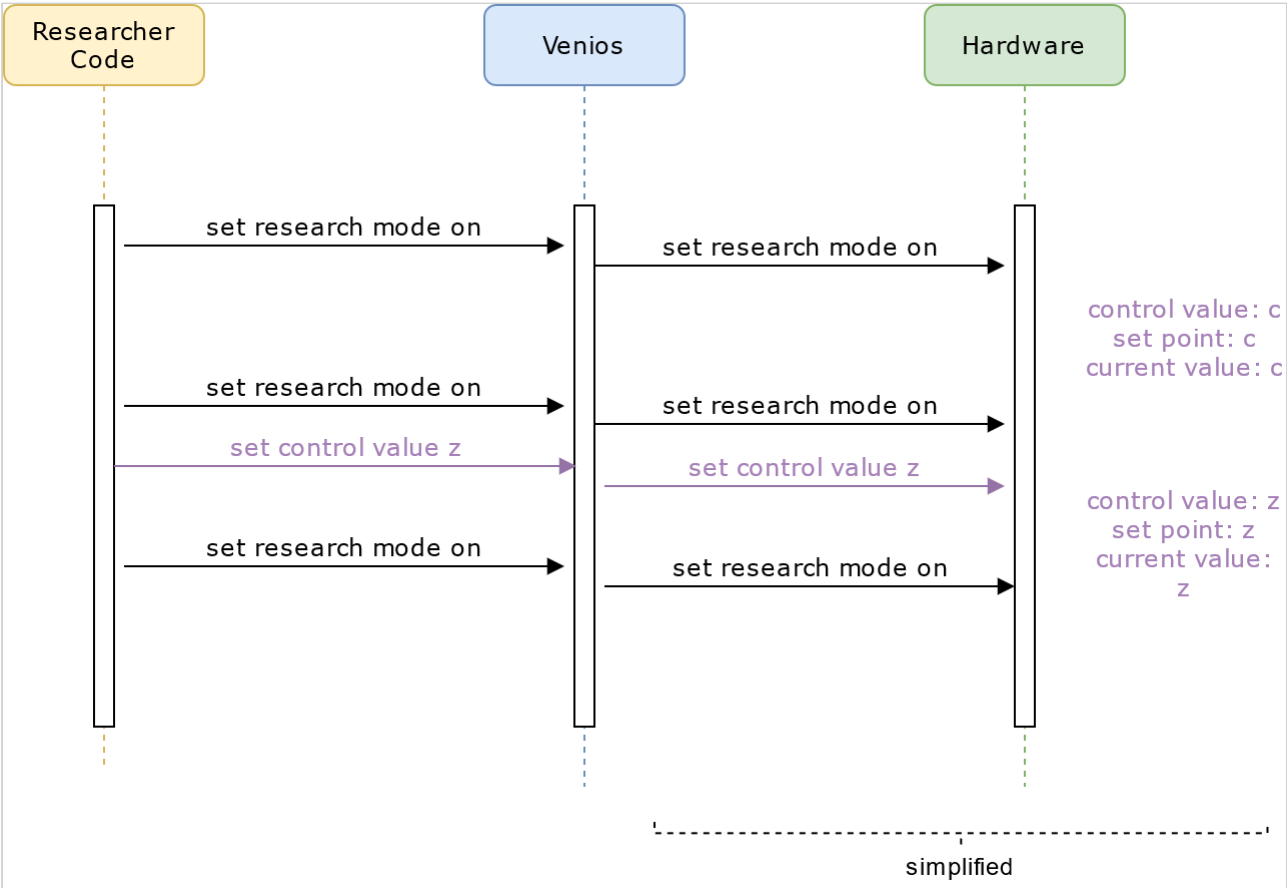


It is important to initialize the control value with the current value / the current state of the hardware, before we start to request control access.
As soon as we have control access, the last set control value will be written to the machine. If we don't set the control value first, it can have an arbitrary value.

```
# initialize all control signals to their corresponding signals to avoid value jumps
initialize_all_control_signals(self.venios_api, all_control_signals)
```

Keep Research Mode On

To get control access (and to keep it), set_research_mode_on needs to be send. This is needed to be able to send new control values, but also so that the last set value is kept and not changed to a fallback value from the hardware itself.



The class ControlAccess continuously sends the research_mode_on-event:

```
# start control access
control_access_uuids = list({control_signal.control_access_uuid for control_signal in
 all_control_signals})
control_access = ControlAccess(
    venios_api=self.venios_api,
    list_of_control_access_uuids=control_access_uuids)
control_access.start()
```

It needs to be stopped at the end of the experiment:

```
control_access.stop()
```

Check Control Access is Granted

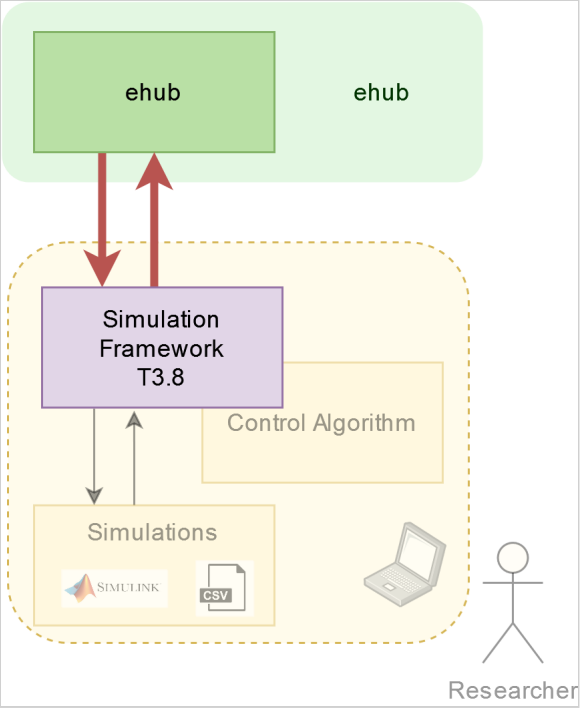
After initializing control access, we check if control access has been granted by ehub/ESI

```
# wait until control access flags are set (at most 10 requests)
```

```
ready &= wait_for_control_access(self.venios_api, all_control_status_signals,
experiment_uuid)
```

2.1.4 Python: ehub-Connection

In order to connect the Simulation Framework to ehub directly (without using the Venios platform), Python code - following the design of the SFW - has been implemented within the repository of SCS. This page provides basic documentation of its usage.



2.1.4.1 Details

https://github.com/ReMaP-CH/remap-controller-example/tree/master/controller/common/ehub

The code will be integrated in the SFW directly. Here the most important details:

- interface_ehub.py[13]
  - InterfaceEhub
    - class for read and control
    - *start_communication_with_HW()*
      connects to ehub and sets up reading and control
    - *data_exchange_ehub()*
      must be executed to read and control
    - *stop_communication_with_HW()*
      ends the connection to ehub
    - *input_signals*
      can be set to set control values (for all signals configured as **CONTROL**)

13 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/ehub/interface_ehub.py

        - *output_signals*
          can be accessed to read the most current value (for all signals configured as **READ** or **READ_CTRL_ACCESS**)
        - Watchdog-Signals are send in their own thread, every 2 seconds.
    - InterfaceEhubReadOnly
        - class for read only (will not send any control values)
        - *start_communication_with_HW()*
          connects to ehub and sets up reading
        - *data_exchange_ehub()*
          must be executed to read
        - *stop_communication_with_HW()*
          ends the connection to ehub
        - *output_signals*
          can be accessed to read the most current value (for all signals configured as **READ** or **READ_CTRL_ACCESS**)
- opcuaclient_subscription.py[14]
  Ehub-Client provided by Empa. Should/Can be updated, when Empa provides a new version.

2.1.4.2 Tests/Examples

- test_ehub_control_example.py[15]
  shows how a control-loop can be implemented
- ehub_access_with_logging.py[16]
  shows how logging can be configured in main
- test_ehub_api.py[17]
  a few tests on the functionality, also using directly the ehub-client (without InterfaceEhub)

2.1.4.3 Signal Configuration

Example Signal Configuration

ehub_control_example_signals.csv[18]

opc_ua_path	type	initial_control
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.rIstwert	READ	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.bAckResearch	READ_CTRL_ACCESS	

14 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/common/ehub/opcuaclient_subscription.py
15 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/test/ehub/test_ehub_control_example.py
16 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/test/ehub/ehub_access_with_logging.py
17 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/test/ehub/test_ehub_api.py
18 https://github.com/ReMaP-CH/remap-controller-example/blob/master/controller/admin/test/ehub/ehub_control_example_signals.csv

opc_ua_path	type	initial_control
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.bFreigabeResearch	READ_CTRL_ACCESS	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.iStatusResearch	READ	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.bError	READ	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.iStatusFehler	READ	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strWrite_L.strVentile.strY720.bWdResearch	CONTROL_WD	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strWrite_L.strVentile.strY720.bReqResearch	CONTROL_ReqResearch	
ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strWrite_L.strVentile.strY720.rStellwert	CONTROL	ns=2;s=Gateway.PLC1.65NT-06401-D001.PLC1.Dummy.strRead.strVentile.strY720.rIstwert

Specification

- **opc_ua_path** given by Empa, path to connect to the signal
- **type** how the signal is used
  - **READ** value will be read
  - **READ_CTRL_ACCESS** signals to check, if ehub has given full control access. The experiment will not start if one of these signals is false.
  - **CONTROL** control signal
  - **CONTROL_WD** Watchdog signal for control. This signal will always be toggled true/false, to let ehub know, the experiment is still running
  - **CONTROL_ReqResearch** RequestResearch signal for control. This signal is set to true, to let ehub know, we need control access.
- **initial_control** should only be defined for signals of type **CONTROL**. At experiment initialization the read value of this signal is used as initial value for this control-signal.

2.1.5 Setup Development

2.1.5.1 Code

The current version of the code can be found on Github:

https://github.com/ReMaP-CH/remap-controller-example

To get access, send your Github Username to Sabine Proll.

It is recommended to clone the code with git (if you're not familiar with git, I recommend using Gitextensions https://gitextensions.github.io/). This way, you can easily get new versions of the code and also contribute changes yourself.

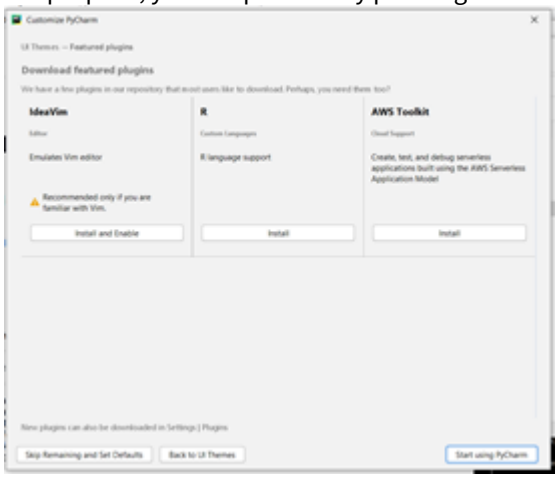
2.1.5.2 Setup

Setup the Python Development Environment

- Install the current Python version from https://www.python.org/downloads/
  On Windows: Install it from the Microsoft Store

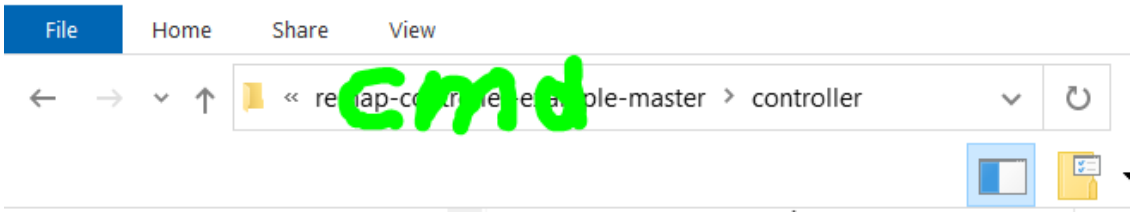
- Install an IDE (integrated development environment), we use PyCharm here (community version, for free), download it from https://www.jetbrains.com/pycharm/download/#section=windows

- During the installation you will be asked to install features as shown bellow. None of them are necessary for this purpose, you can proceed by pressing "Start using PyCharm":



Setup the project in PyCharm

- Download code from github (see link on top of this page)
- Setup a Virtual Environment
  - In folder **controller**/ type "cmd" into the adress line as shown in green bellow:



  - this opens up the command window within this folder and allows you to run python3 from there:

```
python -m venv .venv
```

    Instead of writing "python3" you might have to lookup the path to your python-executable and use it from there. My first command is: "C:\Users\sproll\AppData\Local\Programs\Python\Python38-32\python.exe -m venv .venv" → which is ok, since the python.exe is being copied to the .venv folder. So further executions of the code are not affected.
  - then run the two following commands one after another

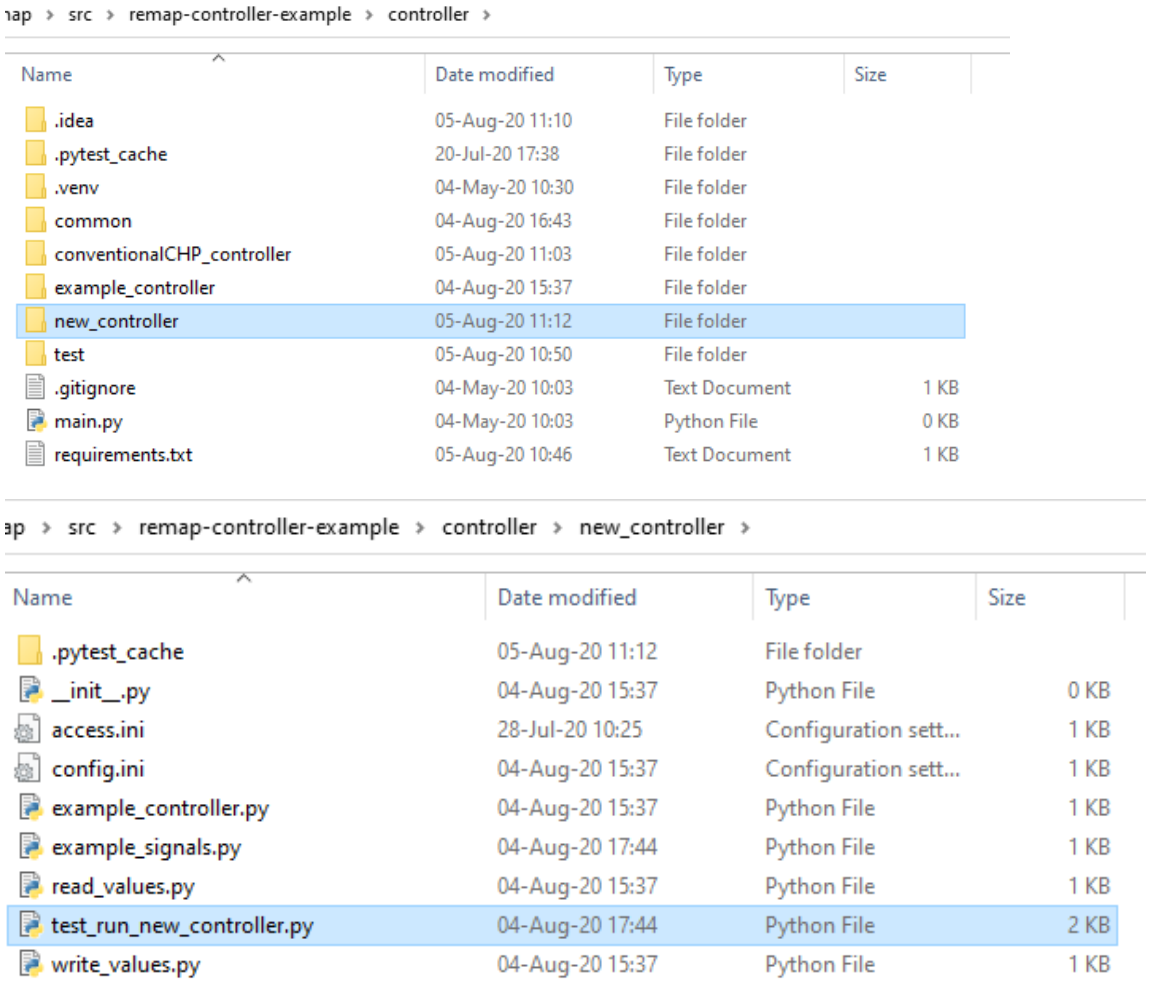
```
.venv\Scripts\activate.bat
pip install -r requirements.txt
```

    See also the original documentation on setting up a virtual environment: https://docs.python.org/3/tutorial/venv.html
    → This works better than setting up the virtual environment with PyCharm!
  - Now you are ready to open the folder **controller/** in PyCharm.
- Possible problems:

- You might need to install certain C++ libraries additionally for some of the Python libraries (depending on the local setup). I installed "Visual Studio Build Tools 2022" with "Desktop development with C++"

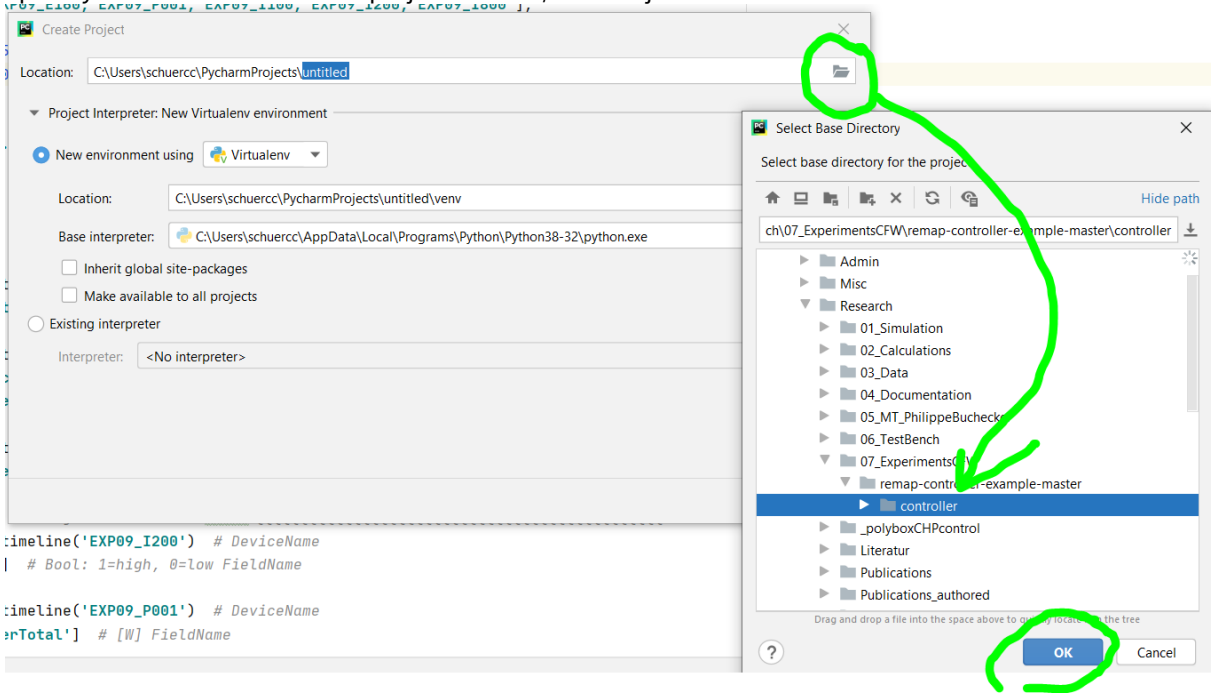
Make your own controller

- Ask Sabine Proll[19] to send you a .zip folder with a controller example code.
- In the folder ***\remap-controller-example-master\controller*** copy the folder ***example_controller\ehub*** or ***example_controller\esi*** depending on your needs and rename it to your liking (in this example it is now called ***new_controller***)
- rename the file ***test_run_example_controller.py*** to your liking.

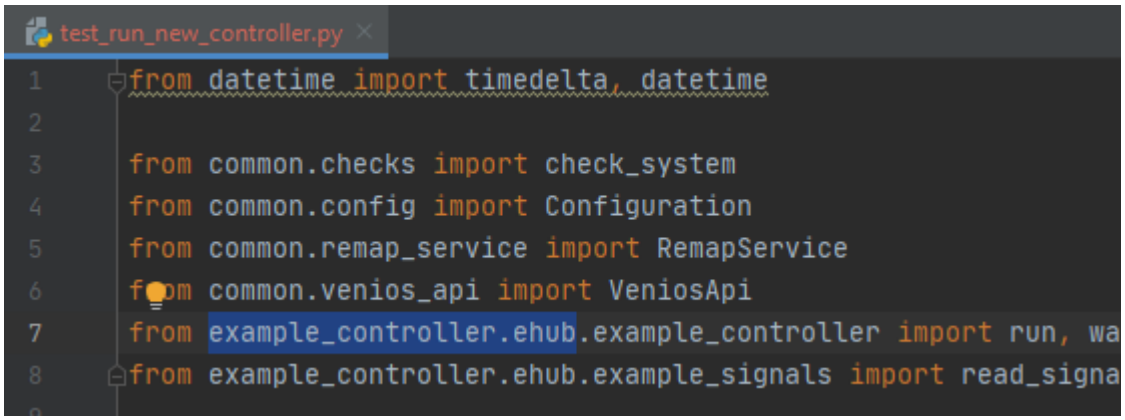


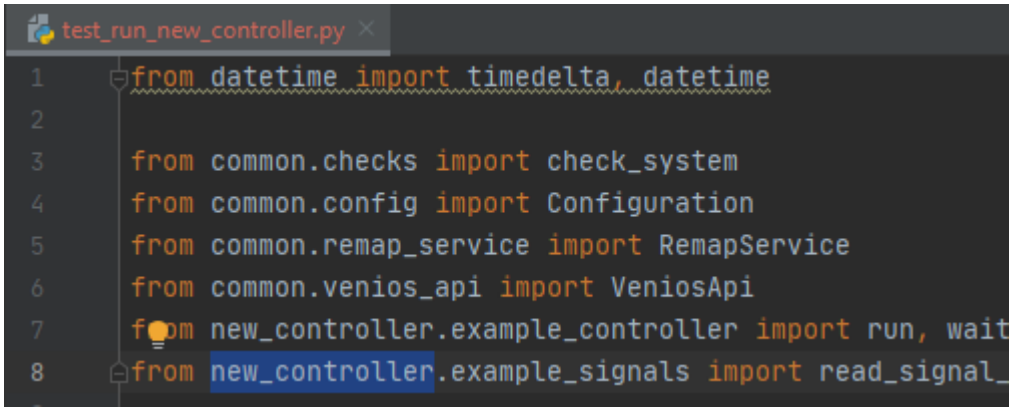
19 https://tools.scs.ch/wiki/display/~sproll

- Open PyCharm and create a new project via "File/New Project". Select the folder "controller" and click "OK".



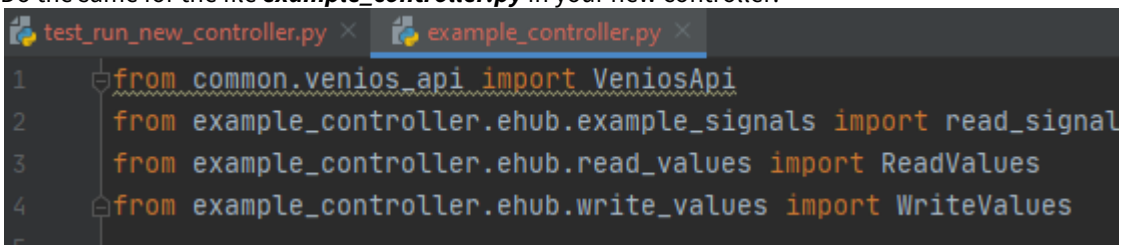
- In PyCharm, navigate to your own new controller and open the file **previously** called ***test_run_example_controller.py***.
- Replace the terms "example_controller.ehub" or "example_controller.esi" with the name of your new controller:

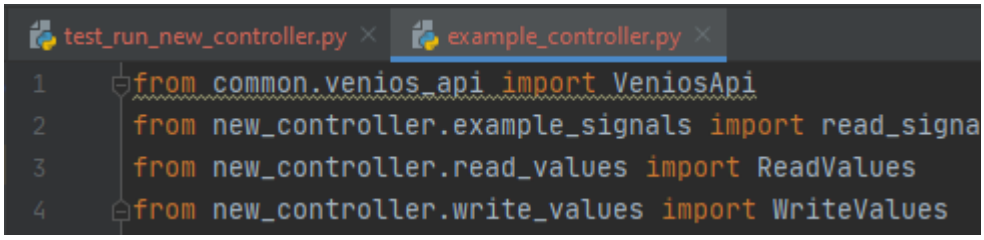




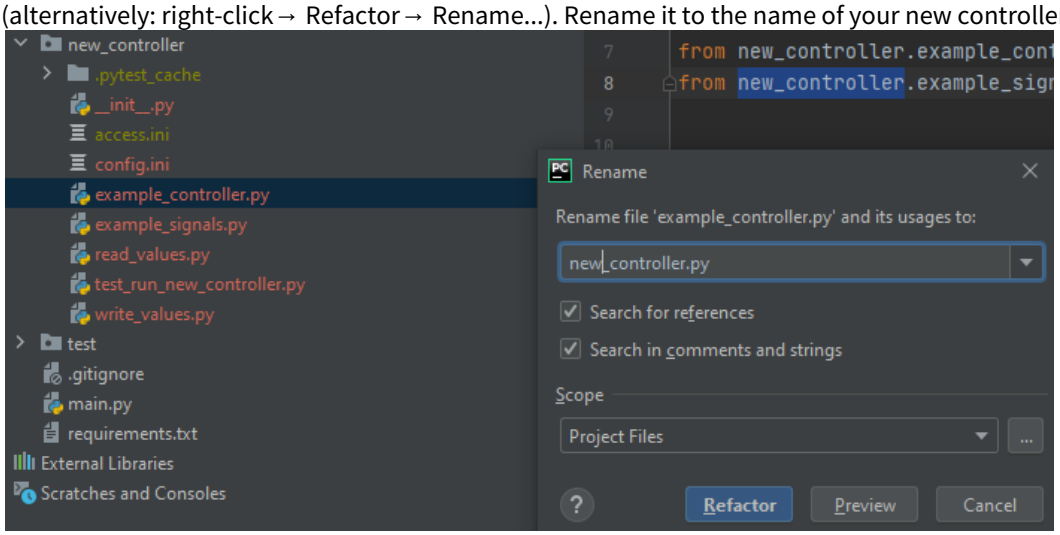
You should not see errors (words underlined in red color)

- Do the same for the file ***example_controller.py*** in your new controller:

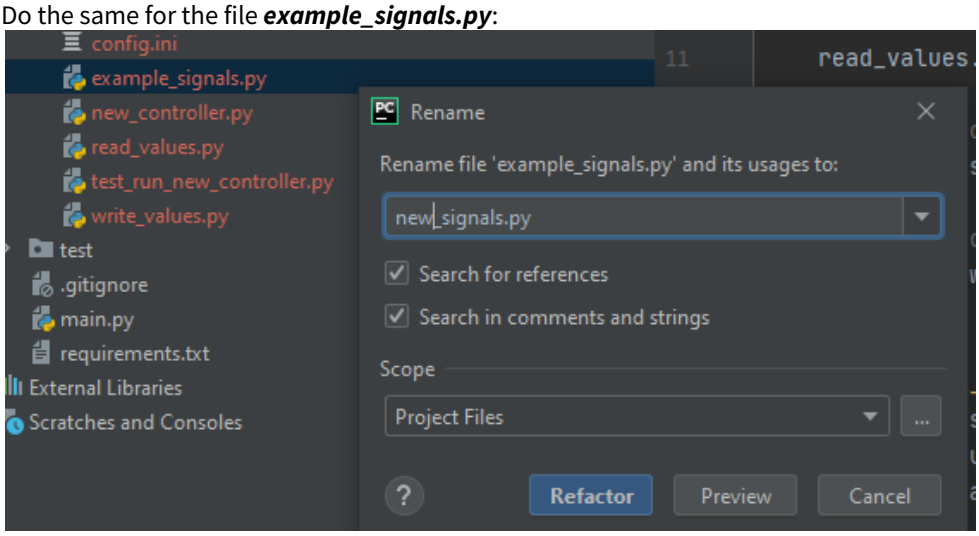




- In PyCharm, select the file ***example_controller.py*** of your own new controller and press Shift+F6 (alternatively: right-click→ Refactor→ Rename...). Rename it to the name of your new controller:



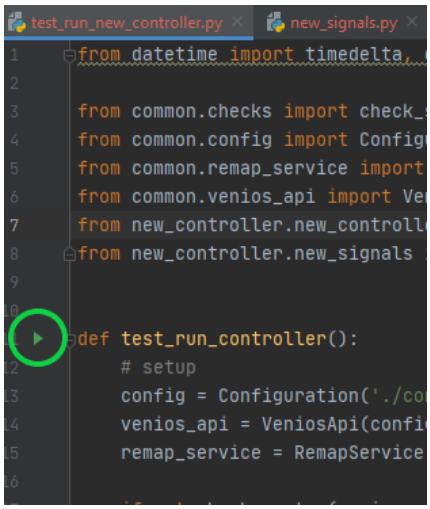
- Do the same for the file ***example_signals.py***:



- Define all signals that you are going to use in the ***new_signals.py*** file. There are two types of signals: ReadSignal and ControlSignal. If a control signal is bound to a read signal, you should define it with "corresponding_read_signal=...", so that its value will be initialized with the correct value before starting the experiment.
- Open the files ***read_values.py*** and ***control_values.py*** and adapt / extend the classes to the needs of your controller code.
- Now change the logic (in your new_controller file) to your liking by using and importing the signals and values you defined in the previous steps.
- Delete / change the assertions to your liking.

```
# assert: that relative position has been switched in den last 40 sec
config = Configuration('./config.ini', './access.ini')
venios_api = VeniosApi(config)

data = venios_api.get_data_timeline('65NT_DUMMY_Y720',
                                    start=datetime.utcnow() - timedelta(seconds=40),
                                    end=datetime.utcnow())

sd = sorted(data, key=lambda d: d['ValueSetTimestamp'])
without_duplicates = [sd[0]]
last_element = sd[0]

for i in range(1, len(sd)):
    if sd[i]['RelativePosition'] != last_element['RelativePosition']:
        without_duplicates.append(sd[i])
        last_element = sd[i]

assert len(without_duplicates) > 2

for i in range(1, len(without_duplicates)):
    if without_duplicates[i-1]['RelativePosition'] <= 50:
        assert without_duplicates[i]['RelativePosition'] == 60
    else:
        assert without_duplicates[i]['RelativePosition'] == 40
```

- Everything is now ready to be tested - good luck! You should be able to start the test by clicking on the green arrow next to it:



Access to Venios

You have to get your own login credentials for Venios → contact Sabine Proll[20] for this.

Code, that needs the credentials, will automatically crash the first time you execute it and create a file "access.ini", where you can add your credentials. After that, the code should execute.

Communication with ehub

The correct behaviour depends on Venios being connected to the ehub. If this is the case, can be checked by running

- test_run_example_controller.py : test_run_controller()

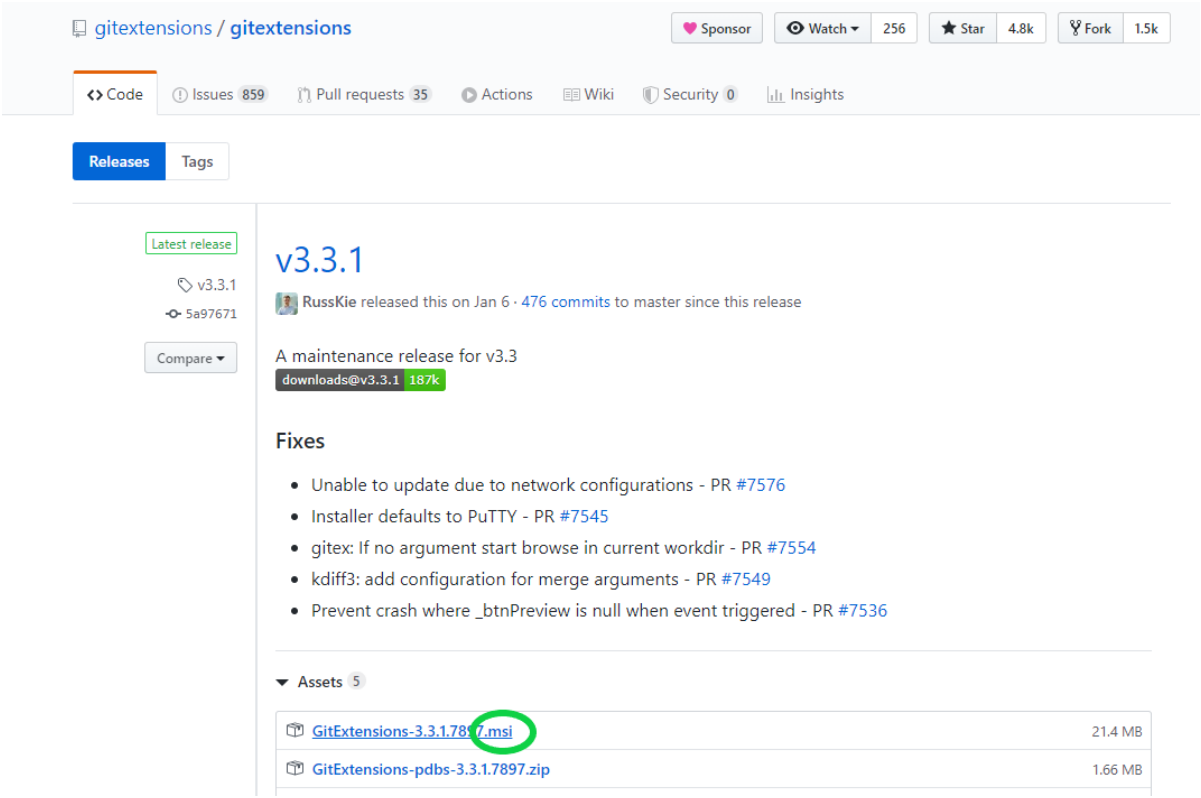
If this test is green, then communication with Venios, all the way to ehub, is working. If not, contact Sabine Proll[21]

2.1.5.3 Working with git

Cloning the project

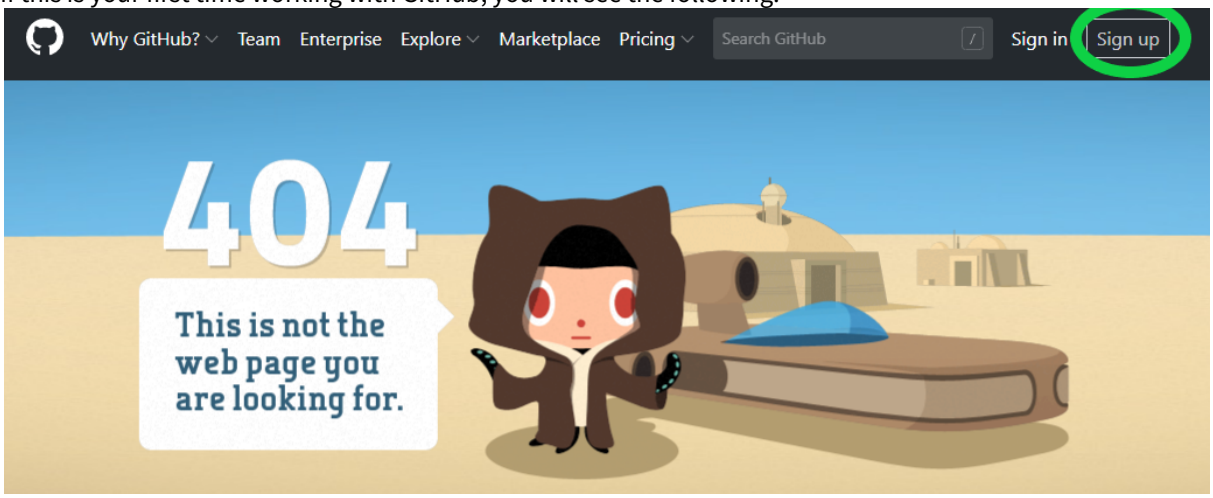
1. install Git, download it from https://git-scm.com/downloads
2. We recommend using gitextensions if you are a beginner: it is a graphical interface for git. The latest version can be downloaded here: https://github.com/gitextensions/gitextensions/releases/latest

20 https://tools.scs.ch/wiki/display/~sproll
21 https://tools.scs.ch/wiki/display/~sproll



3. Run the .msi (windows installer) file and install the program. You can choose all the default options.

4. The repository (where the python code is stored) is located on GitHub: https://github.com/ReMaP-CH/remap-controller-example.
   If this is your first time working with GitHub, you will see the following:

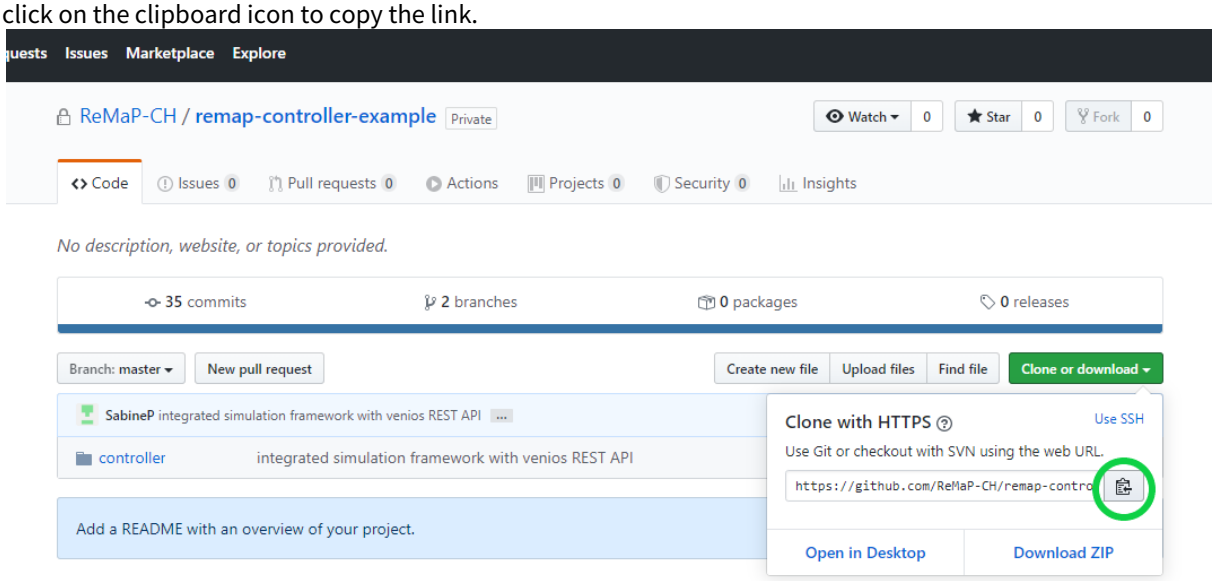


   This is because the project is private. You will need to create an account by clicking on "Sign up".

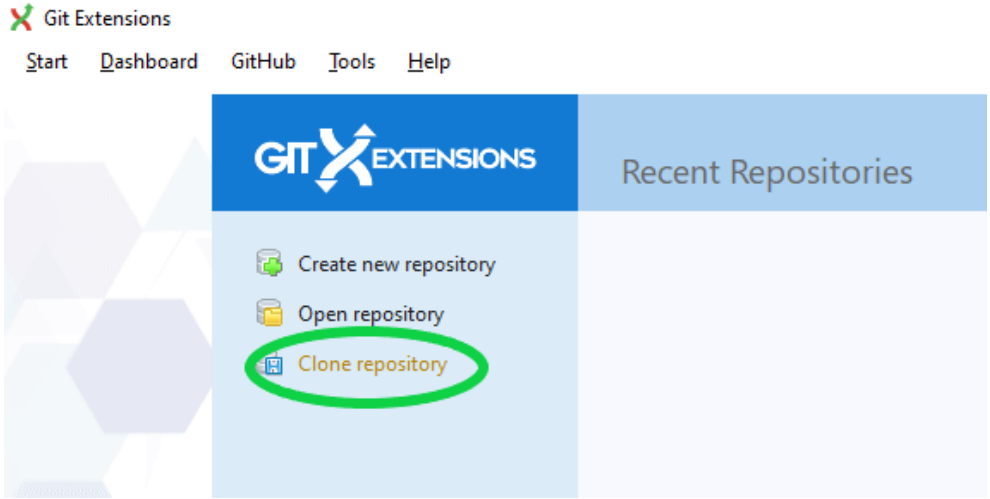
5. Send your GitHub Username to Sabine Proll[22] to get access.

22 https://tools.scs.ch/wiki/display/~sproll

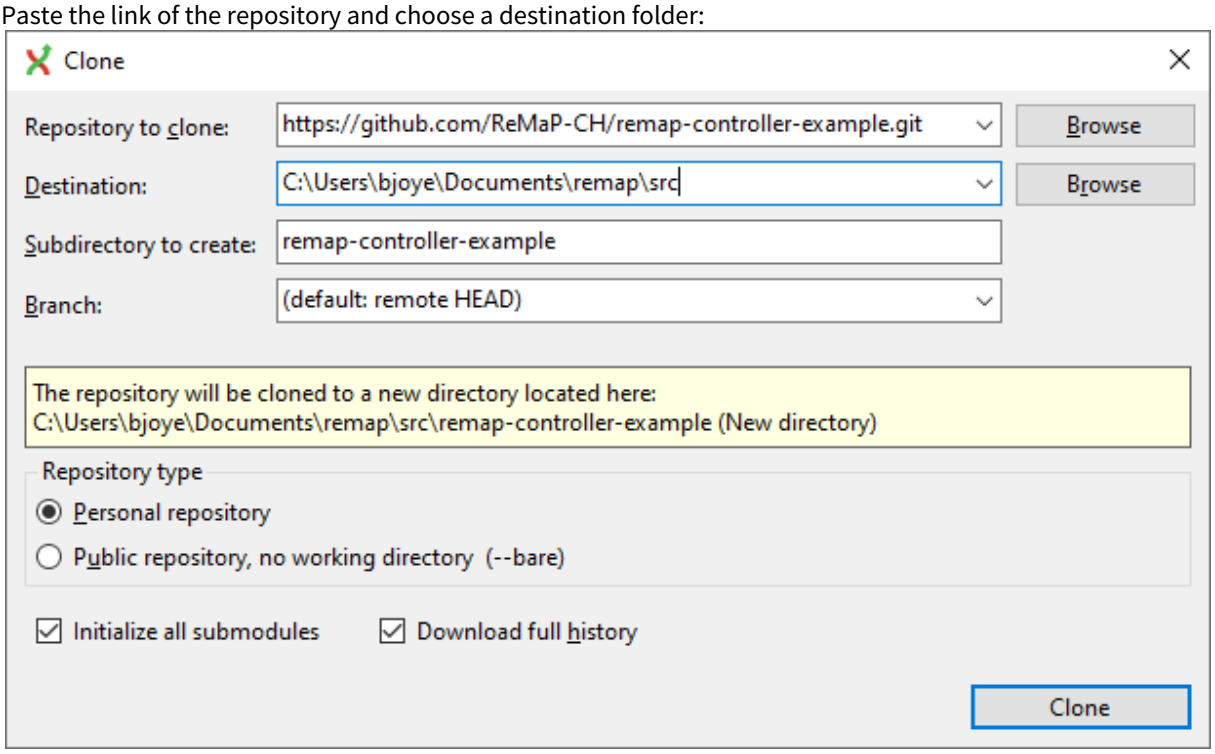
6. Open https://github.com/ReMaP-CH/remap-controller-example again, expand "Clone or download" and click on the clipboard icon to copy the link.



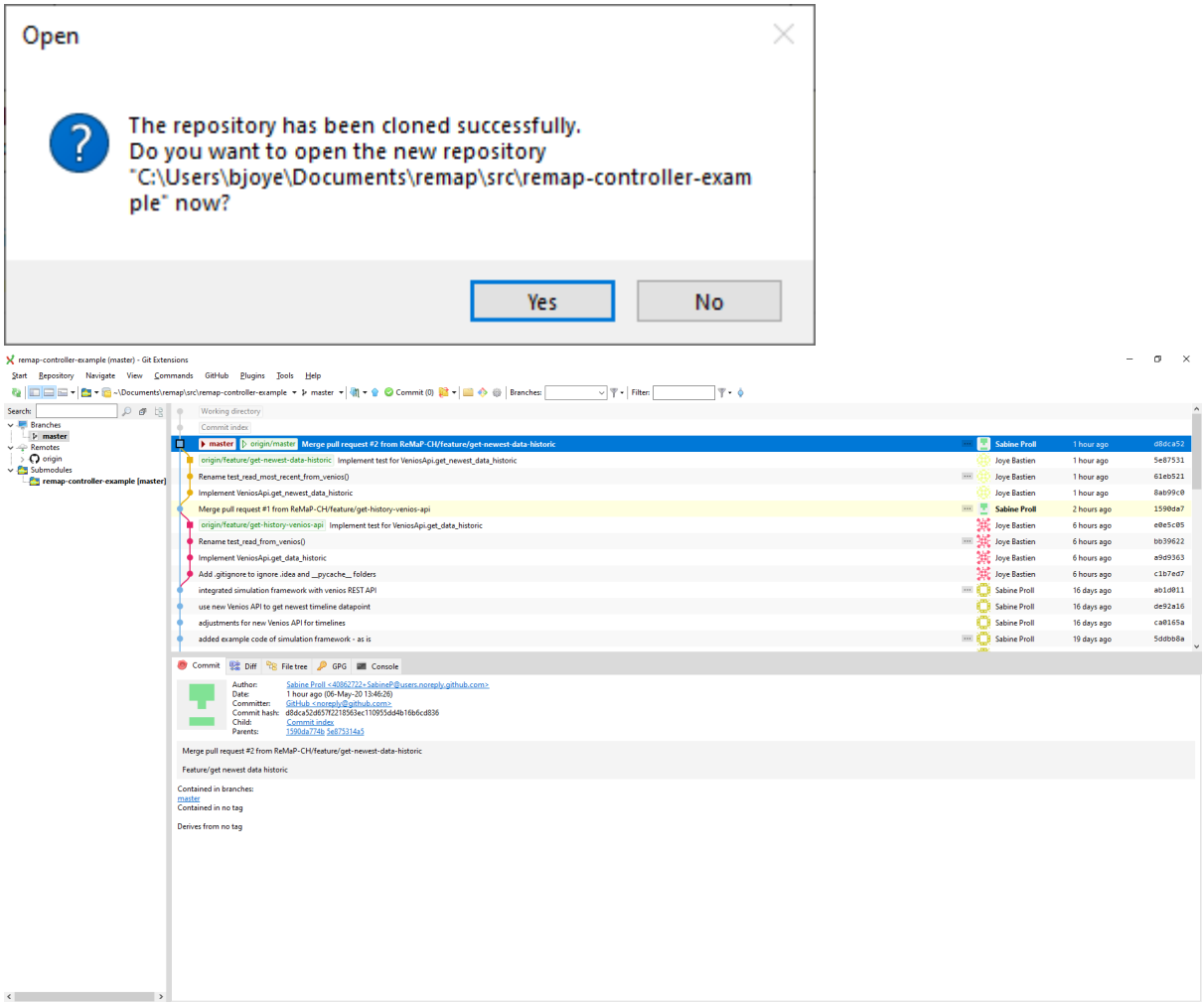
7. Open gitextensions and choose clone repository:



8. Paste the link of the repository and choose a destination folder:

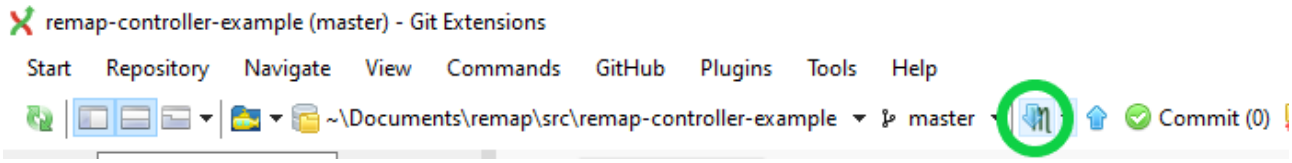


9. Congratulations! You just cloned your first repository! You can now open it.



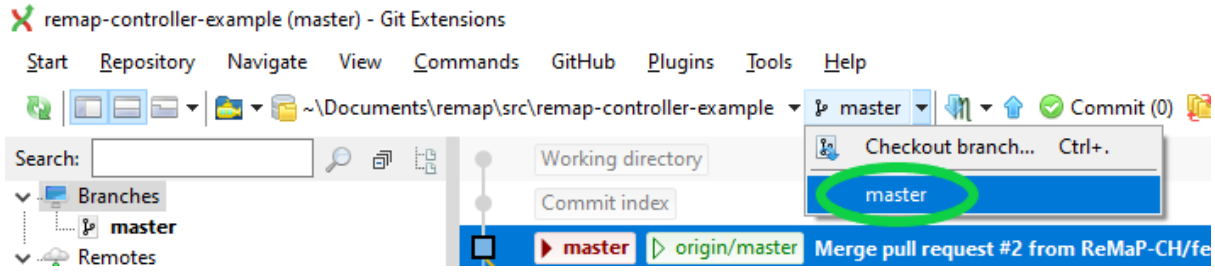
Updating your local version

You can pull the latest version of a remote branch into your local branch.



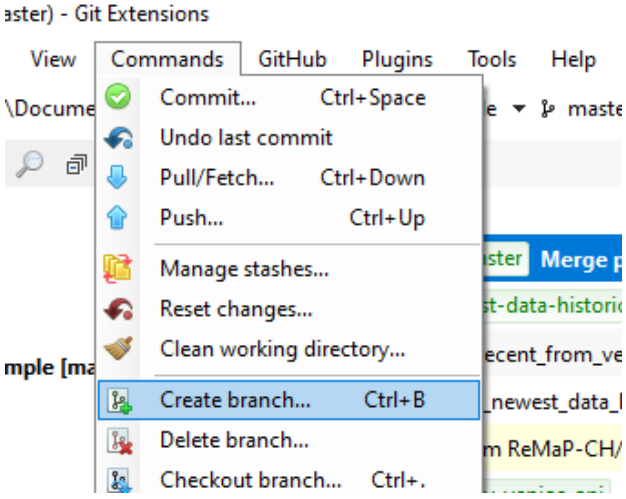
Adding a new feature

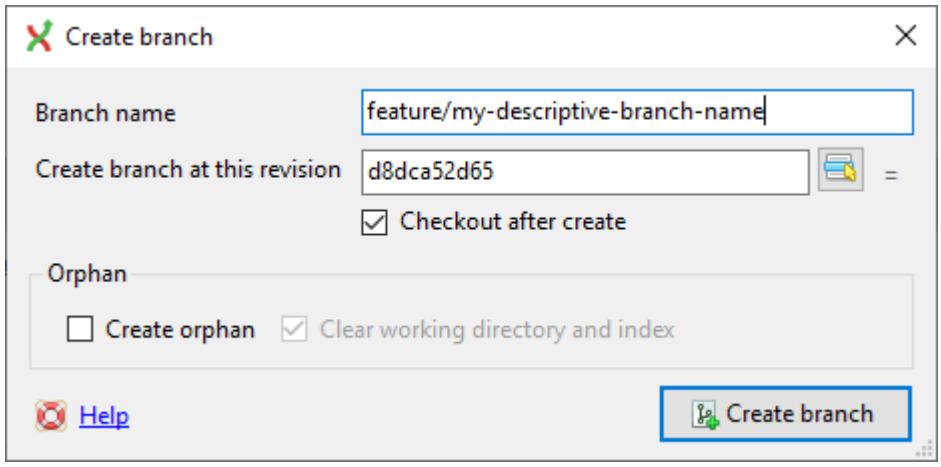
1. Checkout the master branch:



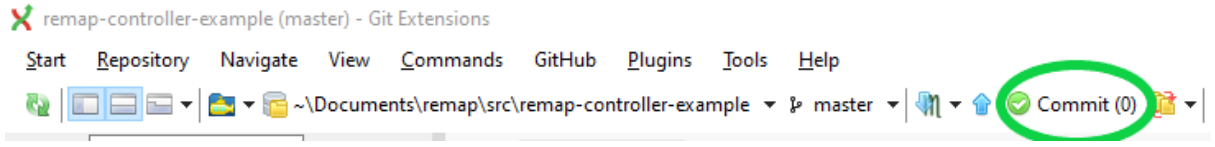
2. Pull the branch to get the latest version (see above "Updating your local version")

3. Create a new branch with a descriptive name starting with "feature/":

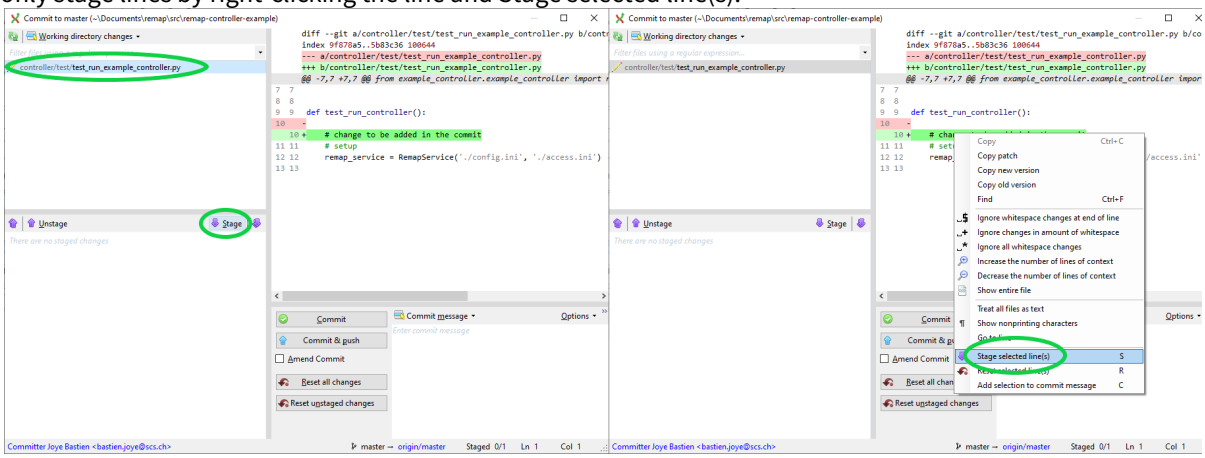




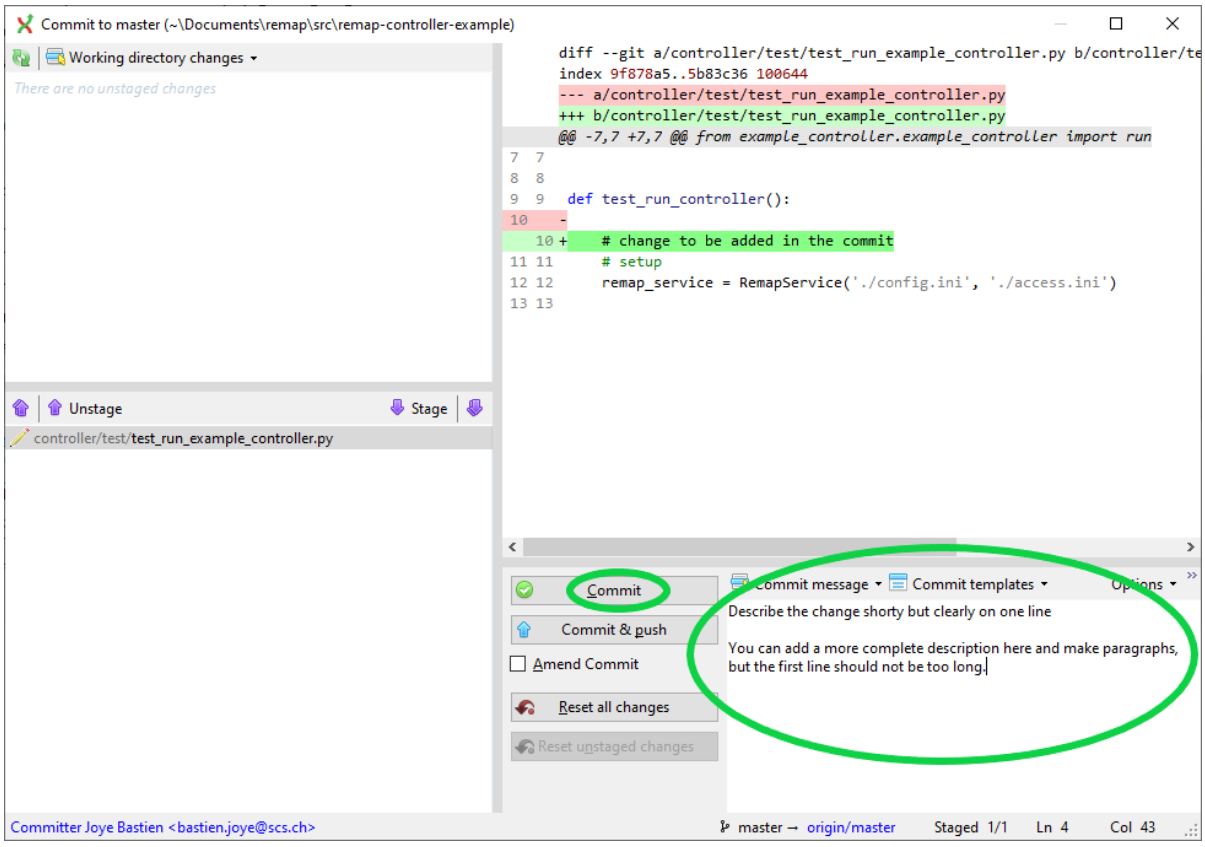
4. Make changes and commit them:



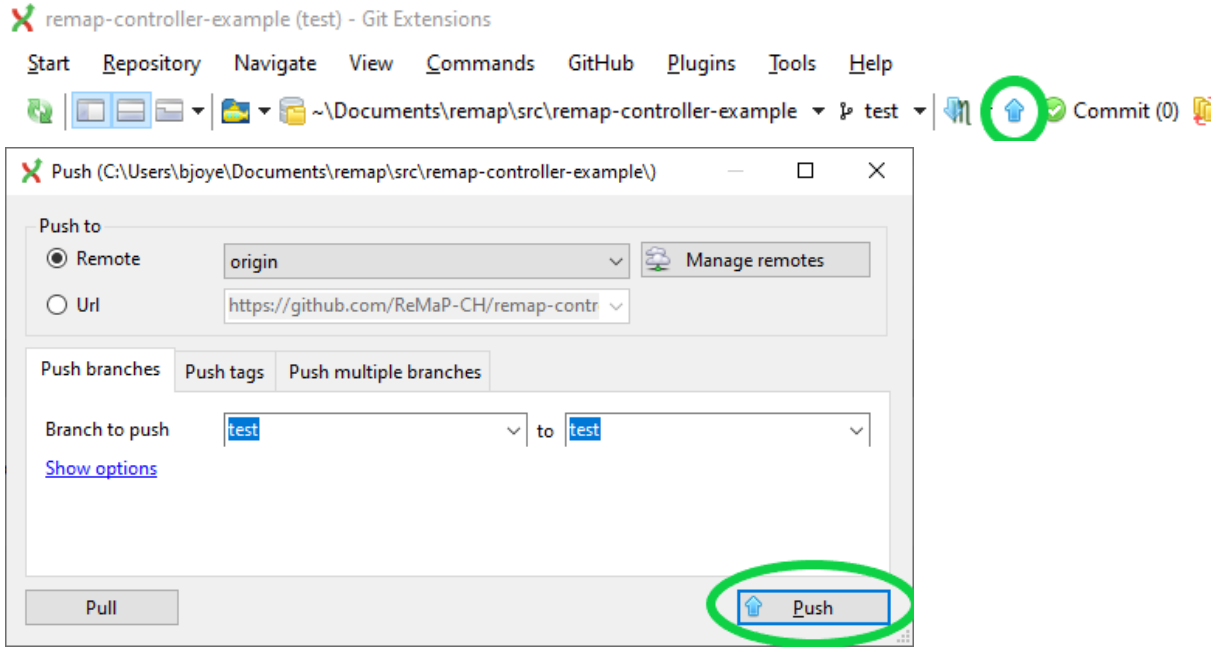
You can see all the files that you modified in the top left panel and their corresponding changes in the top right panel when the file is selected.
First select which files you want to stage (to add them in the commit) and then click on stage. You can also only stage lines by right-clicking the line and Stage selected line(s).



Then write a commit message and click on Commit:

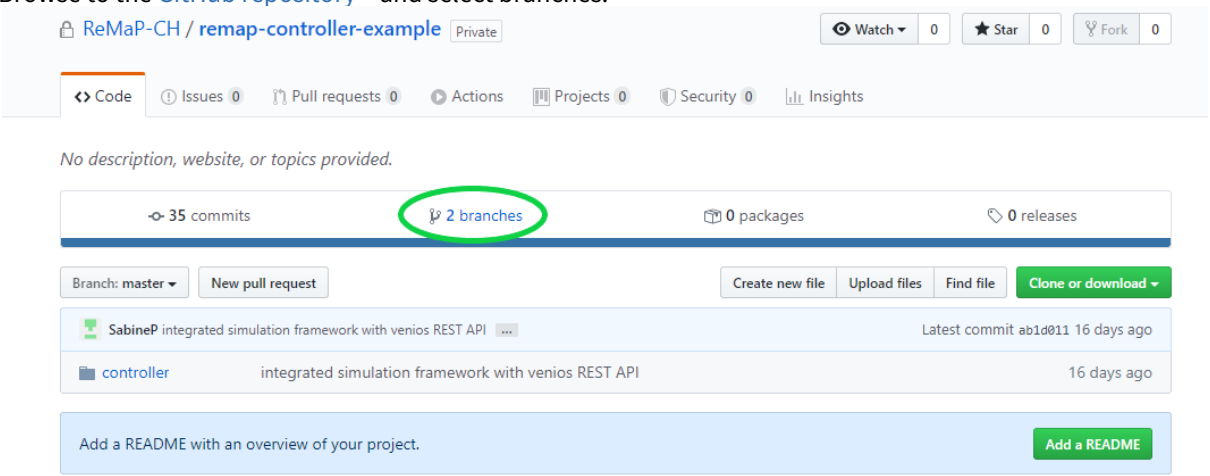


5. Push your branch to the remote repository:



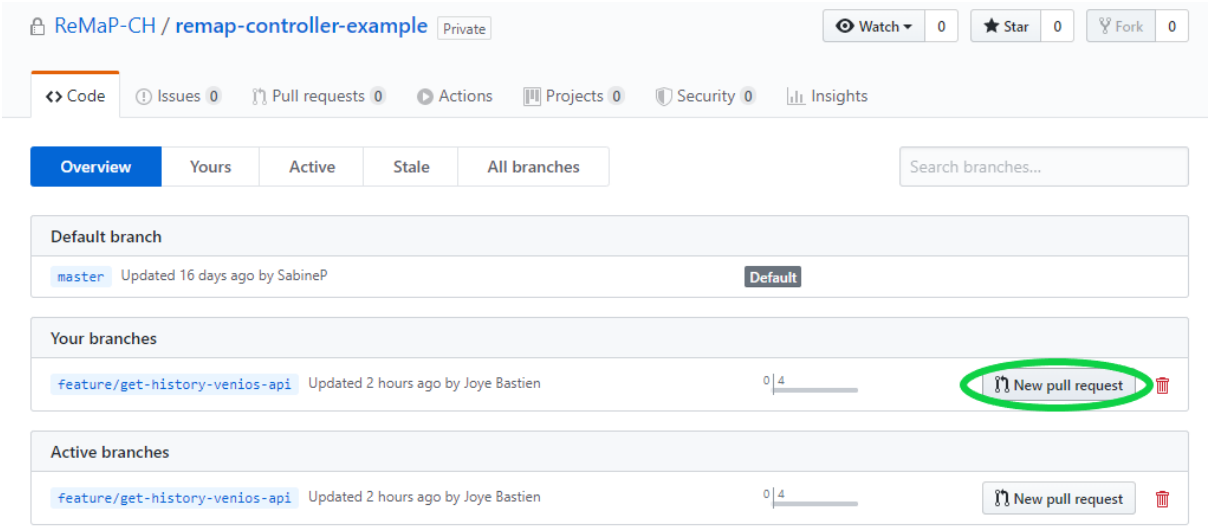
Select Yes for both pop-ups if it is the first time you push this branch.

6. When you are satisfied with all your changes and want your features to be part of the project (in other words, part of the master branch), you can open a pull request for the code owners to review.
Browse to the GitHub repository[23] and select branches:

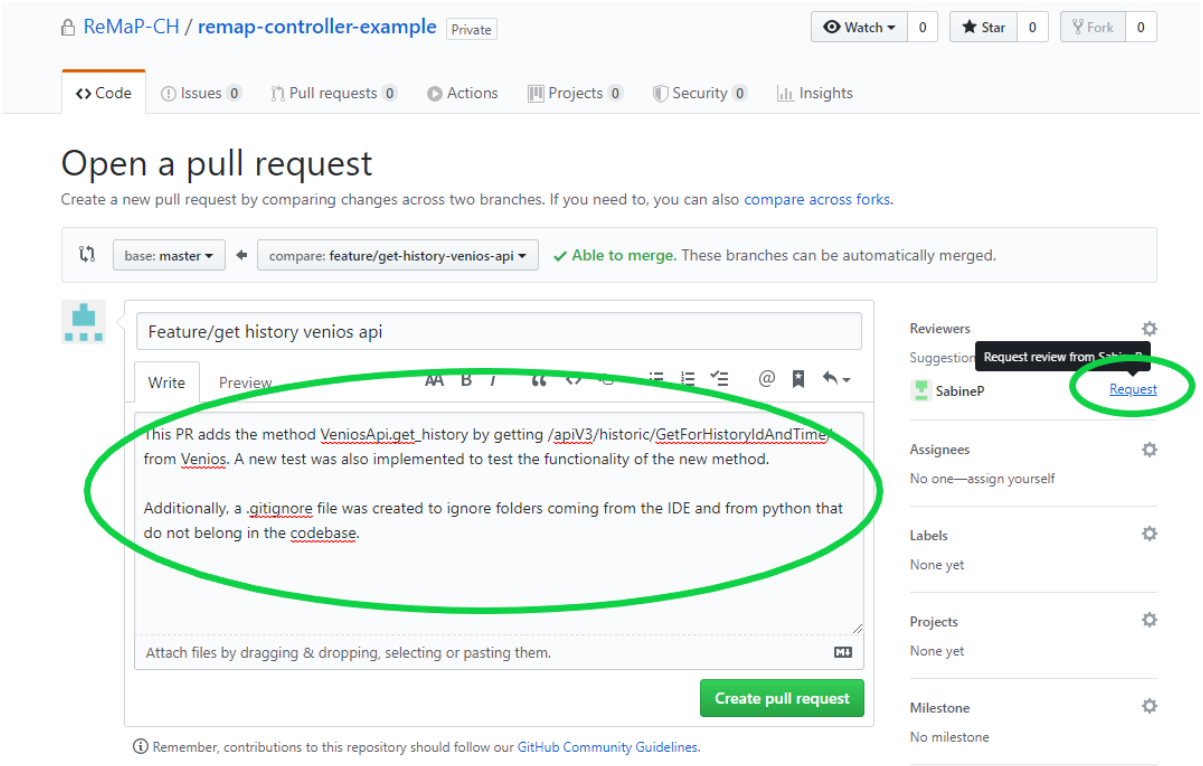


Then click on New pull request next to the desired branch:

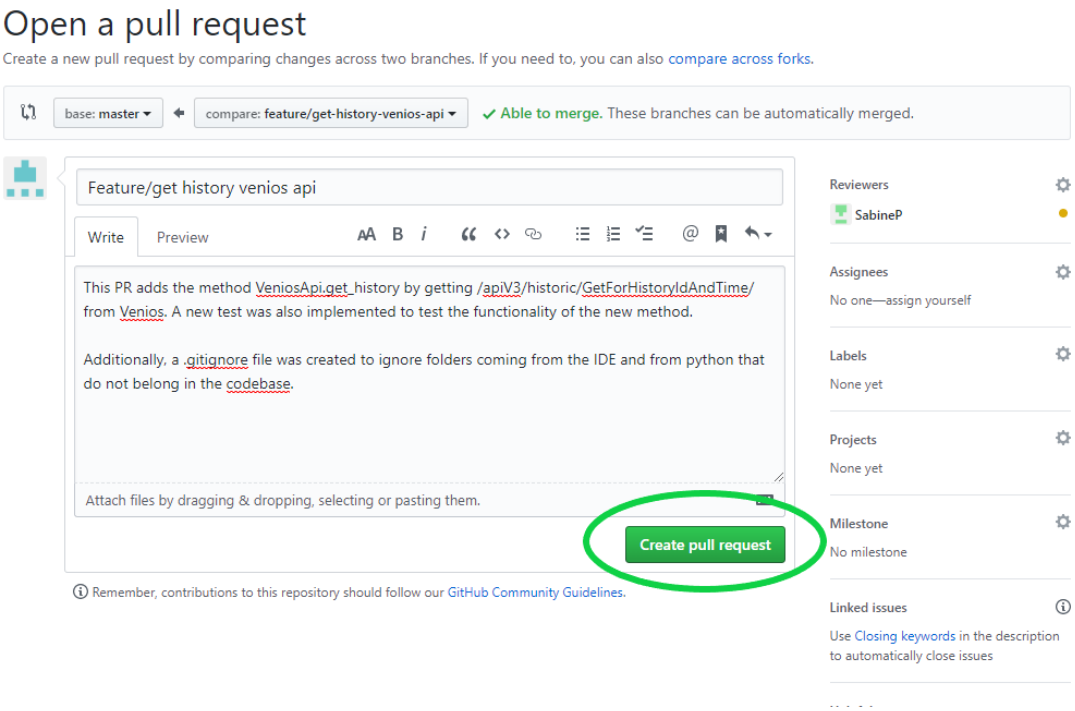
23 https://github.com/ReMaP-CH/remap-controller-example



Add descriptive information about your changes and choose the request the relevant people to review your work:

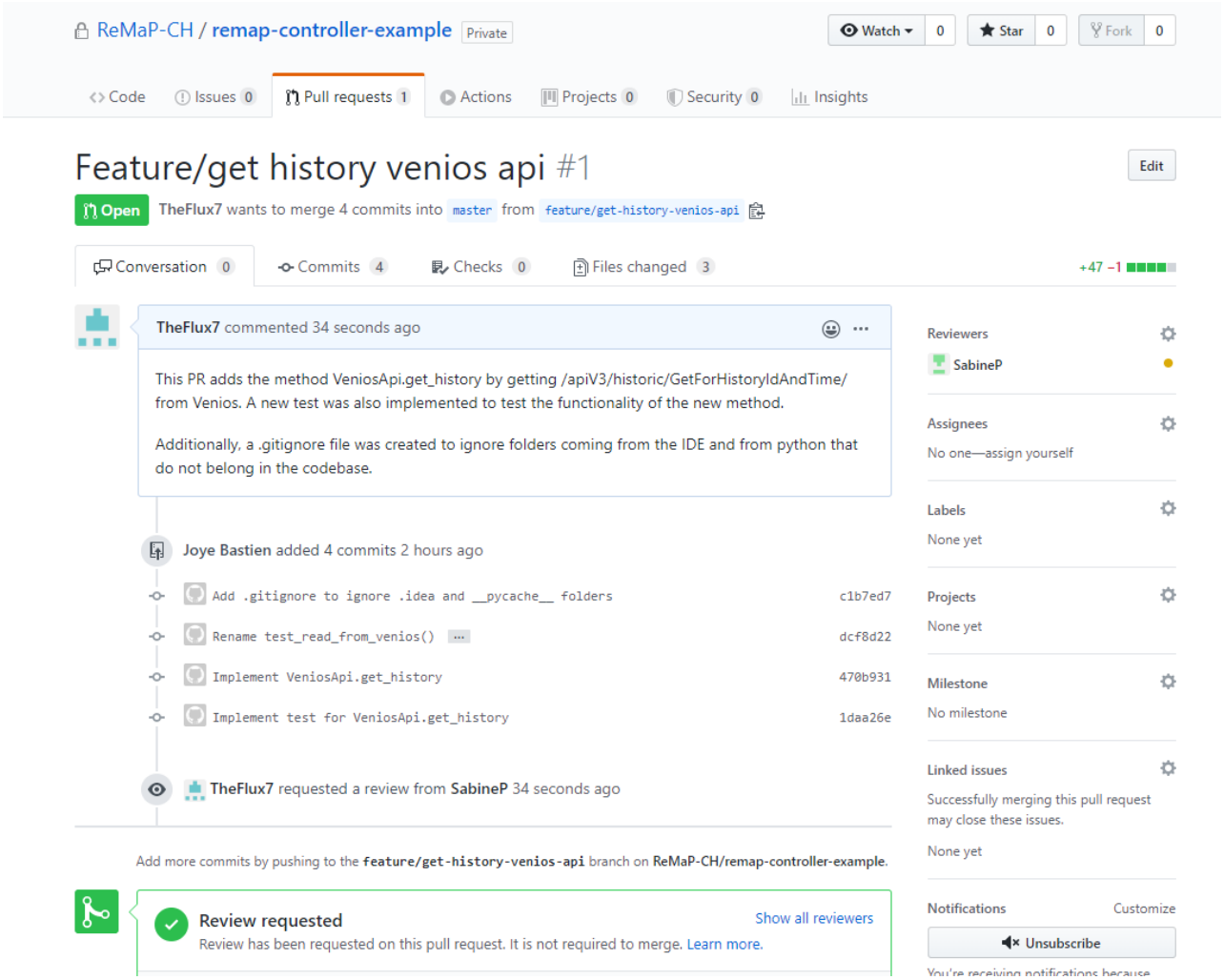


Finally, click on Create pull request:

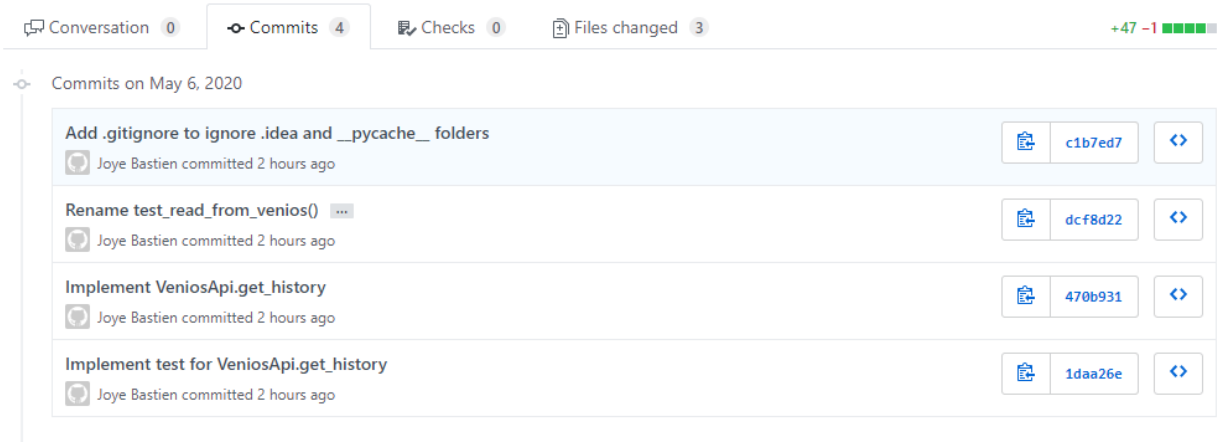


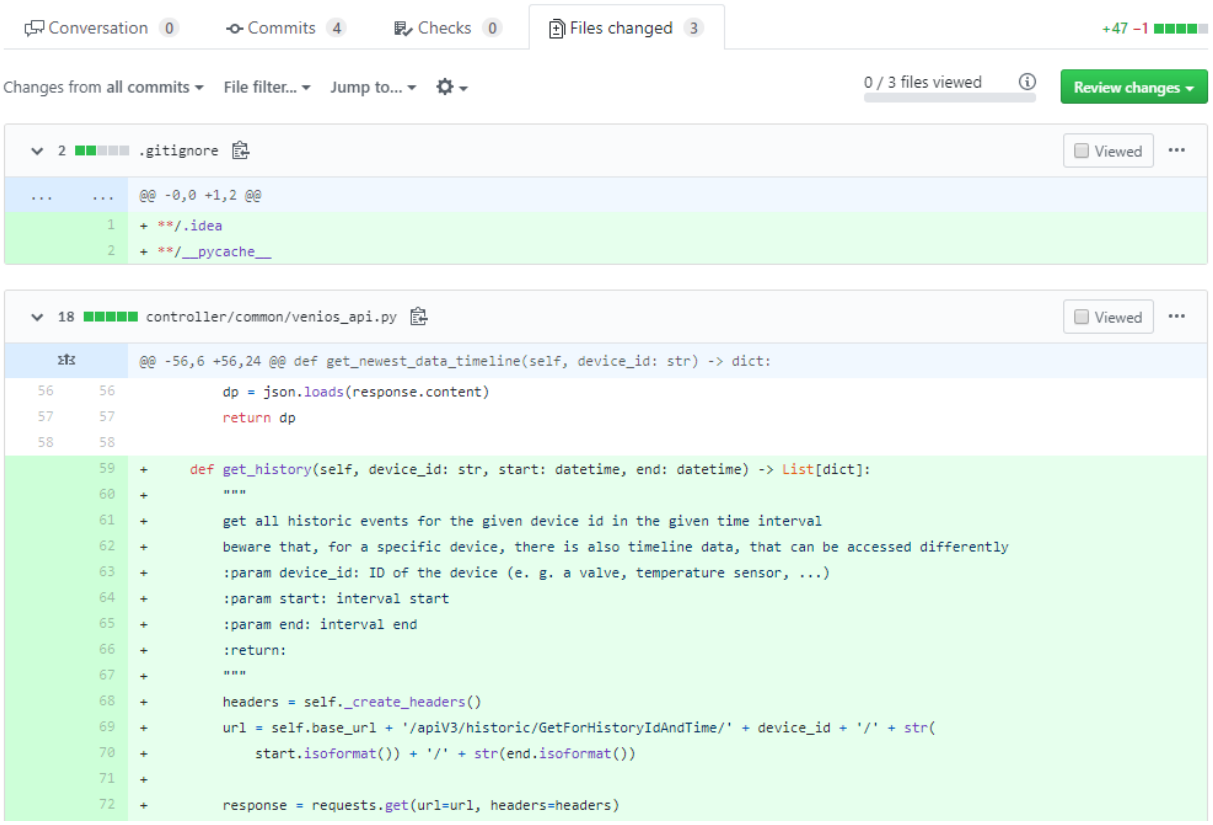
Working with pull requests

An open pull request looks like this:



You can look at the commits, or the changes with respect to the master:





You will receive emails when the reviewer makes comments about your code.

Reviewing is an iterative process, so you will have to commit further changes (see point 4. of Adding a new feature), push them again (see point 5. of Adding a new feature) and notify the reviewer.

Once you fulfill all the requests made by the reviewer, you are able to merge your branch.

Congratulations, your changes are now part of the project!

Background Information git

Useful Git Tools

See slides by Christian Lang[24]: git_tools.pdf[25].

Git Workflows

Motivation
A git workflow defines the way of working with git on a specific project. It is important for the team to work on a common ground, as it **allows consistency and enhances productivity**.

Git is very flexible, and there are many ways to work with git. A great article is available here: https://www.atlassian.com/git/tutorials/comparing-workflows. The article is summarized in the following points.

Centralized workflow

24 https://tools.scs.ch/wiki/display/~clang
25 https://tools.scs.ch/wiki/download/attachments/100436793/git_tools.pdf?api=v2&modificationDate=1627484285630&version=1

No other branches than `master` are required. Each developer commits on `master`.

Advantages

- Simple
- Similar to SVN

Disadvantages

- does not scale with team size
- no mandatory reviews
- broken code in `master` possible

Feature branch workflow
Each feature gets its own branch, so that they can be developed independently. A **pull request** needs to be opened to incorporate the changes into the main `master` branch.

This is the workflow recommended by GitHub: https://guides.github.com/introduction/flow/.

Pull Request (PR)
Reviews by means of PR **improve code quality and team knowledge**, as said in this presentation: PR guidelines (git_and_pr.pdf[26]) by Christian Lang[27].

Merge strategy
GitHub: https://help.github.com/en/github/administering-a-repository/about-merge-methods-on-github

This article provides a good comparison: https://medium.com/@elliotchance/comparison-of-merging-strategies-in-github-2f948c3b8fdc

Merge commit
This is the default strategy: all commits from your branches are kept, and an additional commit merging with `master` is created.

Advantages

- easiest
- history never modified
- branch information kept

Disadvantages

- complex history, not really useful

Squash and merge
The commits of your PR get summarized into a single commit. This commit is then fast-forward merged, which means only one commit is added in total.

Advantages

- linear history

Disadvantages

- commit granularity loss
- history changed

Rebase

26 https://tools.scs.ch/wiki/download/attachments/100436793/git_and_pr.pdf?api=v2&modificationDate=1588751151515&version=2
27 https://tools.scs.ch/wiki/display/~clang

All the commits of your PR are rebased on `master`, no merge commits are created. This allows a linear history.

Advantages
- linear history
- detailed commits

Disadvantages
- history changed

2.1.5.4 Working with venv (python environment)

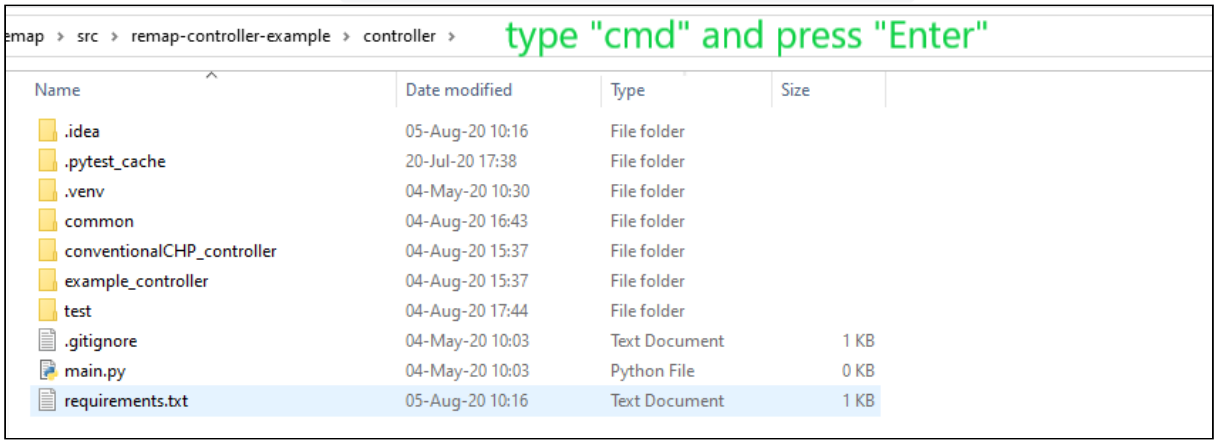
Add and Upgrade Packages

You can add/upgrade packages with PyCharm, or with `pip` within the virtual environment.

The `requirements.txt` file contains the list of packages to install. In order to have a repeatable and correct setup, we should freeze the current versions of installed packages, see https://pip.pypa.io/en/latest/user_guide/#requirements-files.

In consequence, **as soon as we add or upgrade a package**, we should also update the `requirements.txt` file. This can be done easily by following these steps:

1. open a console in the location `remap-controller-example\controller` :



2. activate the environment by typing `.venv\Scripts\activate` in the console.

3. update the `requirements.txt` with the command `pip freeze > requirements.txt`

```
C:\Users\bjoye\Documents\remap\src\remap-controller-example\controller>.venv\Scripts\activate

(.venv) C:\Users\bjoye\Documents\remap\src\remap-controller-example\controller>pip freeze > requirements.txt

(.venv) C:\Users\bjoye\Documents\remap\src\remap-controller-example\controller>
```

4. add `requirements.txt` to the git repository by committing

2.2 Simulation Framework

2.2.1 SFW Setup Guide

(This guide describes the setup for use with integrated Simulink models. If the SFW is used without any Simulink models, steps 4, 5 and 7 can be skipped.)

1. Get the latest SFW code from Adamantios Marinakis (amarinakis@ethz.ch[28])
2. Move and unzip the code to a location of your choosing.
3. Download python 3 here: https://www.python.org/downloads/windows/
    a. You need a 64-bit version to be compatible with Matlab. Make sure your Matlab version supports the python 3 release you want to install.
    b. When installing python, select “add python to path”.
4. Use/install a Matlab version, which is compatible with the python 3 release you want to use. (https://ch.mathworks.com/help/matlab/matlab_external/system-requirements-for-matlab-engine-for-python.html)
    a. Your installation needs to include the following toolboxes:
        i. Matlab Coder
        ii. Simulink
        iii. Simulink Coder
        iv. Simulink Desktop Real-Time
    b. Type the following command into your Matlab command line:
        i. *sldrtkernel –install*
        ii. This installs the Matlab real time kernel. You can also find official documentation here: https://ch.mathworks.com/help/sldrt/ug/real-time-windows-target-kernel.html
5. Install the Matlab Engine API: (https://ch.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html)
    a. Type: *matlabroot* (into your Matlab command window)
        i. Copy the output (Matlab root directory)
    b. Open a windows command prompt with administrator rights
        i. Press: [Windows Key] + [R]
        ii. Type: *cmd*
        iii. Press: [ctrl] + [shift] + [enter]
    c. Install API:
        i. Type: *cd "matlabroot\extern\engines\python"*
        ii. Type: *python setup.py install*
        iii. (If there is no output following the second command, you may need to use the full python path instead of “python”: “*C:/users/**YourUsername**/AppData/Local/Programs/Python/Python38/python.exe*”. Depending on your python release, this path may vary.
6. Follow the instructions at https://tools.scs.ch/wiki/display/SGSREMAP/Python+Example+Code(see page 14) to:
    a. Set up a virtual environment (using your 64-bit, Matlab-compatible python release) in the folder containing the SFW code.
    b. Open the folder as a PyCharm project
7. Test run the SFW code. If the simulation fails, you may want to:
    a. Copy the folder: “C:\Users***YourUsername***\AppData\Local\Programs\Python\Python38\Lib\site-packages\matlab” into your folder containing the SFW code.
        i. If there is no subfolder named “matlab” in your python path, the Matlab Engine API did not install properly.
        ii. Try running the SFW again.

28 mailto:amarinakis@ethz.ch

b. Copy the folder: “C:\Program Files\MATLAB\R2020b\extern\engines\python” into your folder containing the SFW code.
    i. Try running the SFW again.

2.3 Adaptricity

Adaptricity is a platform for power grid simulation and planning. In the context of ReMaP, Adaptricity provides REST-APIs for the computation of power flows.

Researchers can draw or upload their grid models in Adaptricity and then run power flows using the Simulation Framework's python interface developed by ReMaP Task 3.8 at the Research Center for Energy Networks (FEN). The SFW allows Adaptricity power flows to be integrated into a modular multi-energy simulation model containing researcher control algorithms, exogenous data, hardware models and hardware-in-the-loop capability.

2.3.1 Getting started

The Adaptricity instance for project ReMaP can be accessed at this URL:

https://adt-104611.adaptricity.com[29]

To work with it, each researcher must create an Adaptricity account. Please contact:

- **Damiano Toffanin.** *Senior Project Engineer.* dtoffanin@adaptricity.com[30]
- **Nicolas Stocker.** *Head of Project Engineering.* nstocker@adaptricity.com[31]

You will receive the initial credentials via a secure channel and you can subsequently change your password.

You will receive two pairs of credentials:

- **Basic Credentials:** set of credential for Basic Authentication. Such credentials are the same for each user. They grant access to the Adaptricity server instance, but not to the software. They can be shared across the members of Project ReMaP.
- **User Credentials:** Provided by Adaptricity after the researcher initial email and password credentials are issued by Adaptricity. User Credentials are personal and must not be shared. Two researchers working on the same data must request two different set of credentials.

For accessing certain simulations, you need a preliminary agreement:

Simulation	Requirement
eHub simulation (EMPA)	Written agreement on data usage with Sascha Stoller. (sascha.stoller@empa.ch[32])

Depending on your role and tasks, you will be assigned to one or more teams. You can then work with the platform either directly in the web-app, or through APIs and the SFW.

To access Adaptricity via the Rest-API or the SFW you also need to generate an API-token on the Adaptricity web-app under: "User/My Profile/Create Token".

For power flow calculations via the SFW, please follow the instructions on how to set up the Simulation Framework found in the SFW Setup Guide(see page 47). Fill in the basic credentials and your API-token into the acces_adap.ini and

29 https://adt-104611.adaptricity.com/#/
30 mailto:dtoffanin@adaptricity.com
31 mailto:nstocker@adaptricity.com
32 http://empa.ch

tkn_adap.ini files you have been provided by the SFW team. Please make sure you do not share or publish these files on GitHub or any other platform once they contain your credentials.

You can now iteratively run Adatpricity power flows via the Simulation Framework by specifying the project name and grid model name (of the grid model you have uploaded to Adaptricity) in the SFW input parameters and use your own control algorithm or exogenous data to specify power injections through temporary loads and generators for each timestep.

2.3.1.1 APIs

A selection of the available API routes are listed and documented at the link below.

API documentation

https://adt-104611.adaptricity.com/api/docs/

Please refer to this link to find routes and learn about their usage. **Advanced method:** you can also open the inspector of your web browser and see in the “Network” tab which API routes are called by our software. This will allow you to replicate everything which is done through the UI via the API.

API token. You can find the API token in "User / My Profile"

API quick-start guide:

2021-09-10 API Quick Start Guide.pdf[33]

2.3.2 Power Flow Engine & Grid Model

Adaptricity provides an engine for single-phase power flow. Grids can be added in three ways:

- **Direct drawing.** Grids can be drawn using the interactive grid editor.
- **Import in Adaptricity XML format.** Upload a grid in the Adaptricity native format.
  - Model description: Grid simulations with Adaptricity - Required grid data.pdf[34]
  - Data scheme: gridDataModel - Scheme Essential.svg[35]
  - XML validation schema: dpgsimgridschema_2.38.xsd[36]
  - Modelling of transformers: Instruction for the calculation of the short circuit voltage and the copper losses.pdf[37]
- **Import in Matpower format.** See 'Matpower Converter'

33 https://tools.scs.ch/wiki/download/attachments/131104998/2021-09-10%20API%20Quick%20Start%20Guide.pdf?api=v2&modificationDate=1631263754921&version=1

34 https://tools.scs.ch/wiki/download/attachments/131104998/Grid%20simulations%20with%20Adaptricity%20-%20Required%20grid%20data.pdf?api=v2&modificationDate=1631263075283&version=1

35 https://tools.scs.ch/wiki/download/attachments/131104998/gridDataModel%20-%20Scheme%20Essential.svg?api=v2&modificationDate=1631263075940&version=1

36 https://tools.scs.ch/wiki/download/attachments/131104998/dpgsimgridschema_2.38.xsd?api=v2&modificationDate=1631263075069&version=1

37 https://tools.scs.ch/wiki/download/attachments/131104998/Instruction%20for%20the%20calculation%20of%20the%20short%20circuit%20voltage%20and%20the%20copper%20losses.pdf?api=v2&modificationDate=1631266028965&version=1

2.3.3 Matpower converter

Modelling the grid directly in Adaptricity allows to include geographical coordinates and complex grid components. However, to facilitate the data exchange with researchers, Adaptricity features a converter from-to Matpower[38]. Grids in Matpower format can be directly uploaded to Adaptricity and are automatically converted.

The *Matpower* converter is not meant to be a full-fledged converter: it covers the basic grid elements to simplify import/export. Also, geographical coordinates are not supported by Matpower. The addition of more sophisticated grid components can be carried out directly on the Adaptricity interactive grid editor.

The elements that can safely be imported and exported are:

- 2-winding transformers (without tap)
- Lines
- Static loads (fixed P, fixed Q)
- Static generators (fixed P, fixed Q)

Some of the elements that are not supported by the converter but can be modelled directly in Adaptricity are:

- Shunt capacitors (and any non-static load)
- Voltage regulators
- OLTCs

Find out more about *Matpower* modelling here: https://matpower.org/docs/MATPOWER-manual.pdf

2.3.4 Toy example

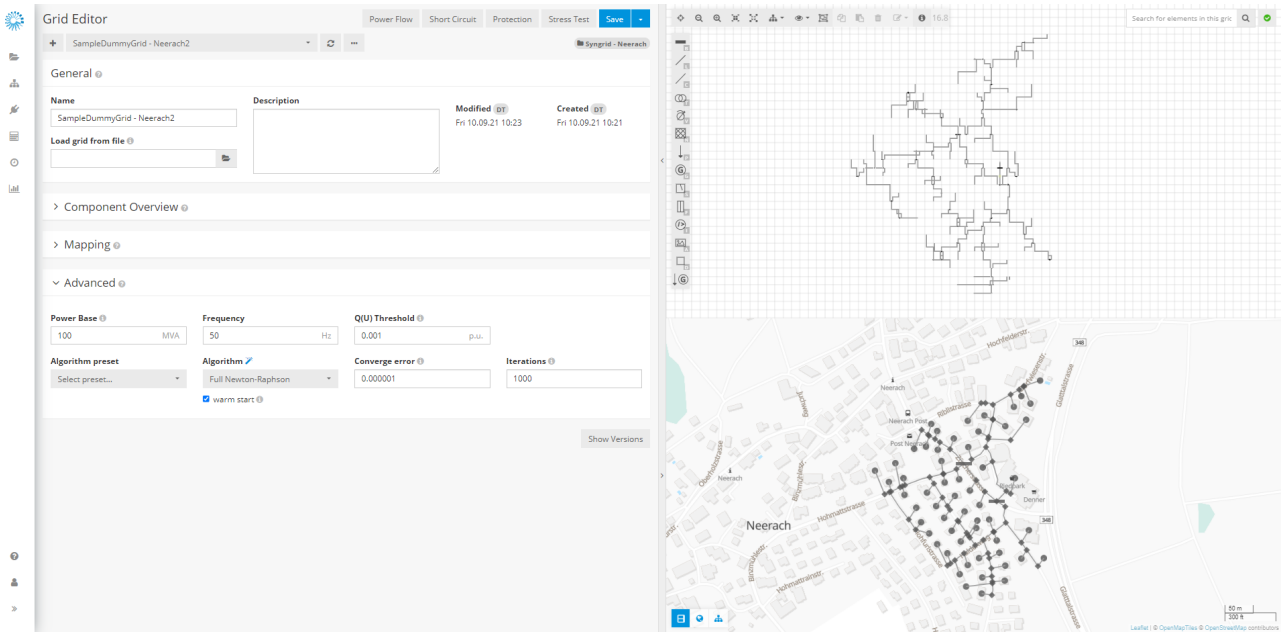
Sample grid in Adaptricity XML format: sampleDummyGrid - Neerach.xml[39]

Sample grid in Matpower .m format: sampleDummyGrid - Neerach.m[40]

38 https://matpower.org/

39 https://tools.scs.ch/wiki/download/attachments/131104998/sampleDummyGrid%20-%20Neerach.xml?api=v2&modificationDate=1631262363894&version=1

40 https://tools.scs.ch/wiki/download/attachments/131104998/sampleDummyGrid%20-%20Neerach.m?api=v2&modificationDate=1631262364182&version=1



2.3.5 Frequently Asked Questions

2.3.5.1 APIs

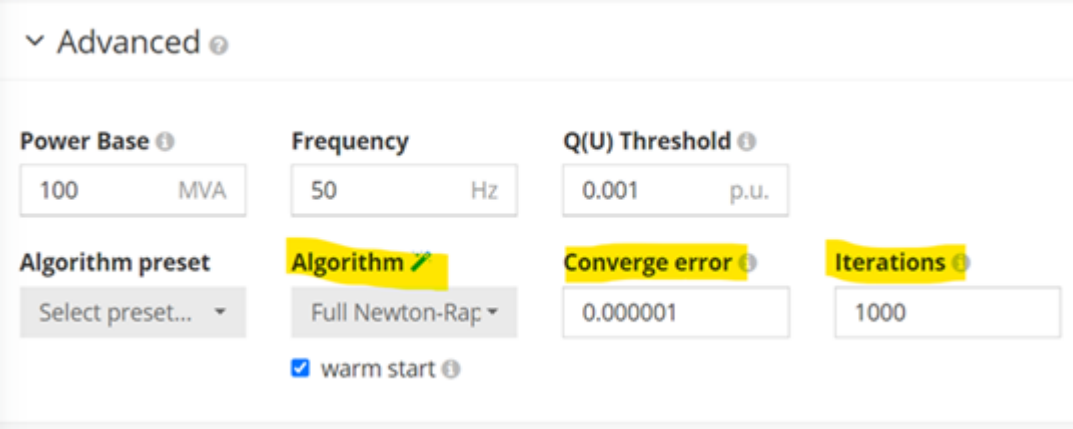
If I specify loads and generators in the input body of the API call, do these replace the loads already defined in the grid model on Adaptricity or do they coexist?

Loads and generators are indeed added to the existing ones, not replaced. Any load or generator that is directly included in the grid model will still be considered.

The power-flow results obtained from the GUI do not exactly match the ones I get from the API call. Why is that?

The Power Flow options `Algorithm`, `Tolerance` and `MaxIter` are set independently in the API and the Adaptricity “Power Flow” page.

If you ever want to compare the results of the PF from the API with the ones from the PowerFlow page, such options must be set independently to the same values. To set them one the webapp, see Section “Advanced” in the “Grid Editor” page (pic below).



2.3.5.2 Grid models

Is it possible for me and another researcher to access the same grid model on Adaptricity, even though we are not part of the same team? Or should we maintain our versions of the model separately?

Each team has a completely separated database, so unfortunately you cannot access the same grid model.

How do I model transformers?

Check out this document: Instruction for the calculation of the short circuit voltage and the copper losses.pdf[41]

How do I model phase-shifting transformers?

Phase-shift transformers are not supported.

How do I model shunt elements, like capacitor banks?

You can model a capacitor like a Voltage-dependent load and define a reactive-load characteristic Q(U). In the software: add load and set Q(U) as Reactive power. You can define your own curve (see picture below).

Example file: gridExample - Capacitor with Q(U).xml[42]

41 https://tools.scs.ch/wiki/download/attachments/131104998/Instruction%20for%20the%20calculation%20of%20the%20short%20circuit%20voltage%20and%20the%20copper%20losses.pdf?api=v2&modificationDate=1631266028965&version=1

42 https://tools.scs.ch/wiki/download/attachments/131104998/gridExample%20-%20Capacitor%20with%20Q%28U%29.xml?api=v2&modificationDate=1631266618528&version=1

Reactive power characteristic

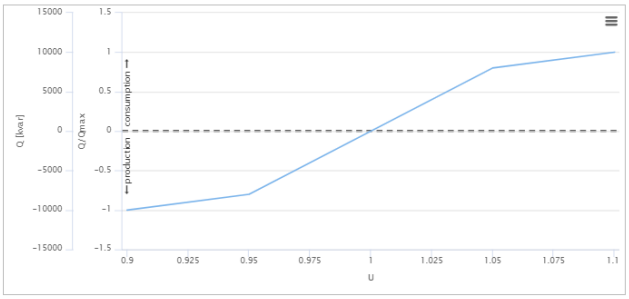
Load load_1

Reactive Power Type

Q(U)

Q_{max}

cos(phi) 0.0995037 ≙ 10000 kvar



U [pu]	Q/Qmax	Q [kvar] (calculated)
0.9	-1	-10000.002
0.95	-0.8	-8000.002
1.05	0.8	8000.002
1.1	1	10000.002

Values are defined in the consumer reference frame

Cancel Apply

3 Hardware Connection

3.1 ehub Connection

3.1.1 Releases for Empa

3.1.1.1 ConnectModule

Newest package

- 12 Jan 2022
  - Installation package: EhubConnectModule_Release_3a27e1ee.zip[43]
  - config.zip[44]

Deployment: see Betrieb ehub-ConnectModule(see page 55)

History:

3.1.1.2 SignallistTranslator

Newest package:

- 19 Mar 2021 ehubSignalListTranslator_Release_bcf40f6a.zip[45]

History:

- 13 Jul 2020 ehub-SignallistTranslator_20200713_selfContained.zip[46]

Verwendung: (siehe auch Integration ehub - Okt 2020[47])

- Im Skript *Run ehub-SignallistTranslator.cmd* sieht man wie der SignallistTranslator verwendet wird.
- Empa kann ein eigenes CSV erstellen und entsprechend das Skript anpassen.
- Fehler werden auf der Commando-Zeile und im Errors.txt-File angezeigt. Weitere Details finden sich im log (im Ordner "log")
- Das File DeviceConfigFile.json ist der Output. Dies ist die Konfiguration für das ConnectModule. SCS kann mit diesem ein ConnectModule starten.

43 https://tools.scs.ch/wiki/download/attachments/110266393/EhubConnectModule_Release_3a27e1ee.zip?api=v2&modificationDate=1641985196597&version=1
44 https://tools.scs.ch/wiki/download/attachments/110266393/config.zip?api=v2&modificationDate=1641991145589&version=1
45 https://tools.scs.ch/wiki/download/attachments/110266393/ehubSignalListTranslator_Release_bcf40f6a.zip?api=v2&modificationDate=1616167327550&version=1
46 https://tools.scs.ch/wiki/download/attachments/110266393/ehub-SignallistTranslator_20200713_selfContained.zip?api=v2&modificationDate=1602491583854&version=1
47 https://tools.scs.ch/wiki/display/SGSREMAP/Integration+ehub+-+Okt+2020

3.1.2 Betrieb ehub-ConnectModule

3.1.2.1 Current package

Siehe: Releases for Empa(see page 54)

3.1.2.2 Configuration

The ehub-ConnectModule can be configured with the following files, all found in the **config** folder:

Config-File	Content
appsettings.json	General settings for ConnectModule, like connection strings, polling intervals, ...
credentials.json	credentials to other systems
DeviceConfigFile.json (generated by SignalListTranslator)	ReMaP-specific configuration: which signals are archived, which signals can be controlled. This file corresponds to the data from the signal list (it is the output of the SignallistTranslator).
Opc.Ua.ehubConnectModule.Config.xml	Configuration for the OPC UA-interface to ehub. The following fields are configured in other files: • URL: appsettings.json • Credentials: credentials.json

3.1.2.3 Log

To analyse problems or to check if the ConnectModule runs properly, check its newest log-file, in the **log** folder.

3.1.2.4 Install, Update & Exchange Config

The ehub-ConnectModule runs with docker-compose. 2 folders are mounted into the docker container:

- /config
- /log

Install

1. Unzip the release package *(e.g. EhubConnectModule_Release_3a27e1ee.zip)* to the place where you want to keep the application
2. In the unzipped release package adjust the local path for the mounted volumes in the docker-compose.yml

```
volumes:
 - /mnt/d/ReMaP/config:/config
 - /mnt/d/ReMaP/log:/log
```

   Replace /mnt/d/ReMaP/config and /mnt/d/ReMaP/log with the corresponding paths in your filesystem.
3. Extract config.zip[48] into the above configured config-folder
4. Fill out the credentials in credentials.json
5. Go to the folder, with the docker-compose.yml and run

```
docker-compose up
```

   In the first run this will result in an error, because of an unknown certificate
6. Copy the certificate in config/certs/pki/**rejected**/certs to config/certs/pki/**trusted**/certs
7. As in 5. run

```
docker-compose up --detach
```

Update

1. Go to the folder with the docker-compose.yml file
2. stop the service with

```
docker-compose stop
```

3. Replace the entire application folder "EhubConnectModule" with the content of the release package
4. start the service with

```
docker-compose up --detach
```

Change config

1. Change configuration file in /config
2. Go to the folder with the docker-compose.yml file
3. stop the service with

```
docker-compose stop
```

4. start the service with

48 https://tools.scs.ch/wiki/download/attachments/126518020/config.zip?api=v2&modificationDate=1641991936195&version=1

```
docker-compose start
```

3.1.3 ehub Signallist Translator

3.1.3.1 Aktuelle Version

siehe Releases for Empa(see page 54)

3.1.3.2 Workflow

Der Signallist Translator generiert das DeviceConfigFile.json für das ConnectModule.

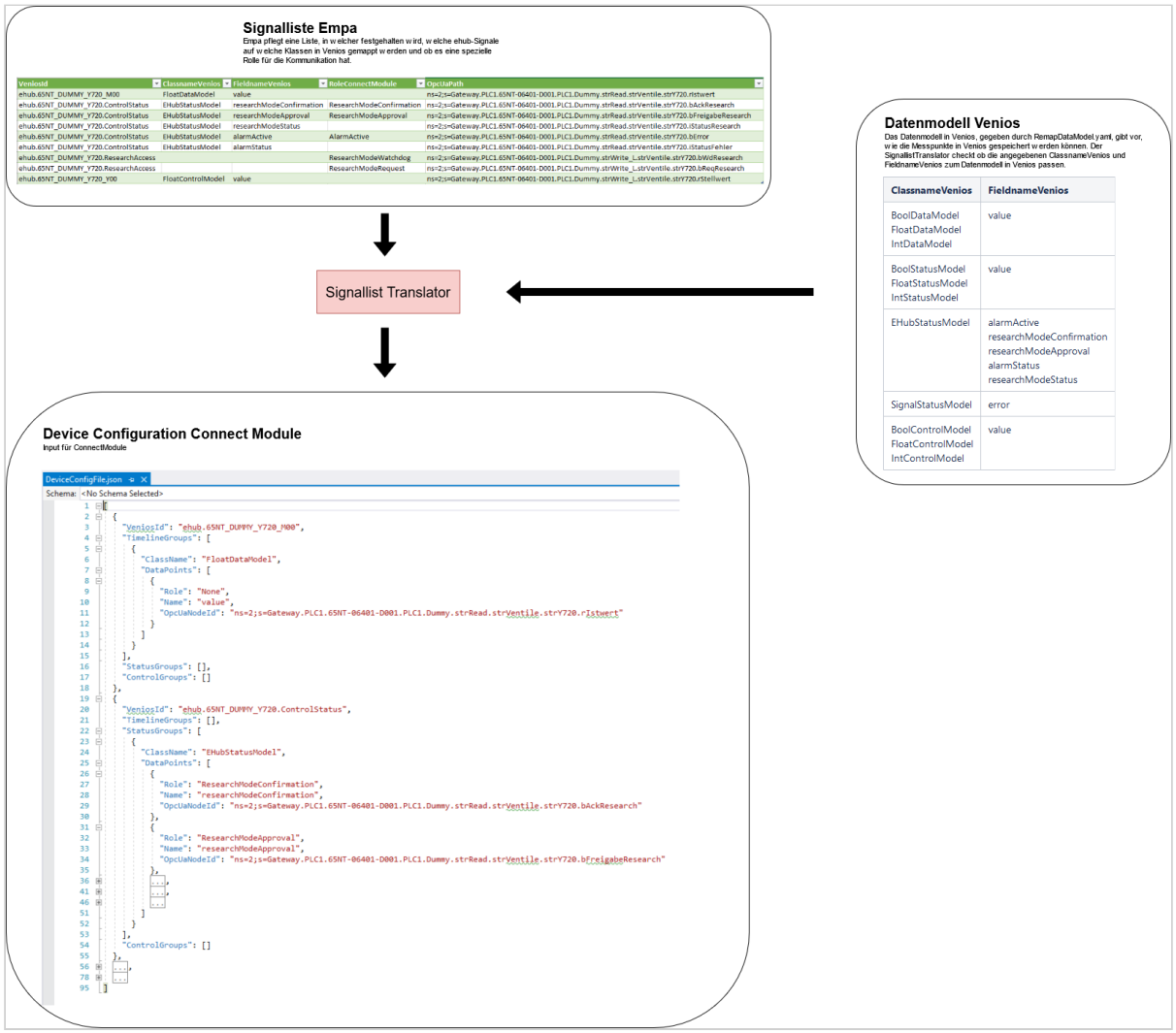
Input files:

- Signalliste.csv[49]
- RemapDataModel.yaml[50] (dies entspricht https://remap.venios.de:8282/swagger/v4/swagger.json)

Output file:

- DeviceConfigFile.json[51]

49 https://tools.scs.ch/wiki/download/attachments/126527476/Signalliste_ehubDummy.csv?api=v2&modificationDate=1629885410430&version=1
50 https://tools.scs.ch/wiki/download/attachments/126527476/RemapDataModel.yaml?api=v2&modificationDate=1629885394349&version=2
51 https://tools.scs.ch/wiki/download/attachments/126527476/DeviceConfigFile.json?api=v2&modificationDate=1629885388074&version=2



3.1.3.3 Constraints

Columns

Column	Description	Restrictions

VeniosId	Identifier used in Venios, e.g., ehub.65NT_DUMMY_Y720_M00	Must be unique within ReMaP. Everything from ehub, should always start with "ehub."
		
		This ID can appear on multiple lines in the signal list.
		
		• The following characters are allowed:
		  ◦ letters: A-Z, a-z
		  ◦ digits: 0-9
		  ◦ special characters: _ .
		• The first character shouldn't be a digit

<table>
<tr><th>Column</th><th>Description</th><th>Restrictions</th></tr>
<tr><td>ClassnameVenios

FieldnameVenios</td><td>Defines the Class and fieldname in Venios, that is used to persist the signal. Classname, Fieldname and VeniosId identify how the data can be accessed in Venios.

The naming of the ClassnameVenios also specifies, how the data is stored:
(...)DataModel
Signal is persisted periodically and when it changes its state
(...)StatusModel
Signal is persisted periodically and when it changes its state
(...)ControlModel
Signal represents a control-value, that can be set over the Venios-REST-API. Signal is persisted periodically and whenever it changes its state.
empty
The signal will not be persisted in Venios.</td><td>If set, then it has to be one of the following combinations of ClassnameVenios and FieldnameVenios (defined in the RemapDataModel.yaml):<table><tr><th>ClassnameVenios</th><th>FieldnameVenios</th><th>Usage</th></tr><tr><td>BoolDataModel
FloatDataModel
IntDataModel</td><td>value</td><td>Measurement data</td></tr><tr><td>BoolStatusModel
FloatStatusModel
IntStatusModel</td><td>value</td><td>Status data</td></tr><tr><td>EHubStatusModel</td><td>alarmActive
researchModeConfirmation
researchModeApproval
alarmStatus
researchModeStatus</td><td>Status of Control-Device (needed to check control access)</td></tr><tr><td>SignalStatusModel</td><td>error</td><td>Error data (only used at PSI)</td></tr><tr><td>BoolControlModel
FloatControlModel
IntControlModel</td><td>value</td><td>Signal represents a control-value, that can be set over the Venios-REST-API.</td></tr></table>
combination VeniosId, ClassnameVenios, FieldnameVenios must be unique.</td></tr>
</table>

Column	Description	Restrictions
OpcUaPath	Full OPC UA path to the signal. This is the same as the NodeId displayed in UaExpert (ns=..;s=Gateway...)	Not empty They are defined by Empa ? Should we check the format "ns=<NamespaceIndex>;s=<Identifier>"?
RoleConnectModule	Field with specific logic in the ConnectModule: • (empty) or None: field has no role • AlarmActive • ResearchModeConfirmation • ResearchModeApproval • ResearchModeRequest • ResearchModeWatchdog	Allowed roles depend on ClassnameVenios: • ClassnameVenios: EHubStatusModel allowed roles: (empty) or None, AlarmActive, ResearchModeConfirmation, ResearchModeApproval • ClassnameVenios: empty allowed roles: ResearchModeRequest, ResearchModeWatchdog • all other Role has to be (empty) or None
~~Samplerate~~	~~Sample interval in seconds. Data is periodically persisted at this rate.~~ ~~(Remark: may be changed to SampleInterval in the future, that is the correct term)~~	Not in use anymore - value will be ignored. Samplerate is configured for all signals globally in appsettings.json of ConnectModule.

Fileformat

- Die Signalliste Empa ist ein File im Format CSV gemäss RFC 4180[52].
- Als Zeichen für die Trennung von Datenfeldern wird das Komma (,) verwendet.
- Sollte ein Feld ein Komma enthalten (was eigentlich nicht vorkommen sollte), muss das Feld mit Anführungszeichen (") umschlossen sein.
- Die erste Zeile muss eine Kopfzeile mit den Spaltenbezeichnern sein.

Das Tool "Signallist Translator" überprüft Restrictions aus obiger Tabelle. Das Tool "Signallist Translator" erzeugt seinen Output nur, wenn es keinen Fehler in der Signalliste Empa findet. Ein Run des Tool wird nicht beim ersten Fehler diesen ausgeben und stoppen, sondern analog wie ein Compiler immer alle Fehler melden.

3.2 ESI Connection

52 https://tools.ietf.org/html/rfc4180

3.2.1 Releases for PSI

3.2.1.1 ConnectModule

Newest package:

- 10 Sep 2021 EsiConnectModuleService_Release_fd28500d.zip[53]
  Changes:
  - Unified sampling: control values are also logged periodically
  - Removed long running call of ReadAllMeasurementDevices (this is only a problem, when there are several 100 devices in the system)
  - Code cleanup: completely removed sampling rate from signals (wasn't used anyway). → old EsiSystemConfigFile.json with SamplingRate should still work (SamplingRate is simply ignored, as before).

History:

- 26 Jul 2021 EsiConnectModuleService_Release_af7af76e.zip[54]
  Changes:
  - Improved logging:
    - log VeniosId, when value is sampled
    - log when exception occurs when writing to OPC UA interface
    - log product version at ConnectModule start
  - Rename executable and dlls to have prefix "ReMaP." (→ update scripts are adjusted accordingly)

  Include git-hash in Version of executable and dlls

- 08 Jun 2021 EsiConnectModuleService_Release_82782083.zip[55]
  Changes:
  - Bugfix ETHESCREMA-2[56]: WebSocket Connection to Venios doesn't recover
    d. h. es sollte jetzt zuverlässiger laufen und die Verbindung für die Steuerung sollte sich nicht mehr aufhängen
- 04 Jun 2021 EsiConnectModuleService_Release_4d333cf1.zip[57]
  Changes:
  - Sampling von Status-Werten ist jetzt gleich wie Sampling von Timelines. Für beides wird die Samplingrate im appsettings.json konfiguriert:

    ```
    "PeriodicSampleInterval": 300000,
    "OnChangeSampleInterval": 1000,
    ```

    Die SampleRate im EsiSystemSettings.json wird jetzt ignoriert!
- 03 Jun 2021 EsiConnectModuleService_Release_99122fe0.zip[58]
  Changes:

---

53 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_fd28500d.zip?api=v2&modificationDate=1631283709835&version=1
54 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_af7af76e.zip?api=v2&modificationDate=1627313380597&version=1
55 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_82782083.zip?api=v2&modificationDate=1623167984393&version=1
56 https://tools.scs.ch/issues/browse/ETHESCREMA-2
57 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_4d333cf1.zip?api=v2&modificationDate=1622813518984&version=1
58 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_99122fe0.zip?api=v2&modificationDate=1622737404653&version=1

    - Bugfix: Control-Signale wurden an falschen OPC UA-Pfad weitergeleitet.
- 22 Mar 2021 EsiConnectModuleService_Release_bcf40f6a.zip[59]
  Vor dem Starten der neuen Version müssen folgende Konfigurationen angepasst werden:
  appsettings.json:
  "VeniosClientUrl": "https://remap.venios.de:8282/",
  "VeniosWebSocketUrl": "wss://remap.venios.de:8282/Apiv4/RemapSocket/OpenSubscriptionSocket/{0}?access_token={1}",
  credentials.json:
  "Venios": {
  "Username": "656123cc-7e95-4148-9221-5e9a01759938",
  "Password": "[KOMMT SEPARAT]"
  },
  Changes:
    - Anbindung an neues Venios V4.0. Insbesondere ist jetzt Steuerung an PSI und Empa gleichzeitig möglich. Rechtemanagement kommt noch, ist aber auch unabhängig vom ConnectModule.
- 11 Nov 2020 EsiConnectModuleService_Release_19e05ff1.zip[60]
  Changes:
    - Bugfix: Connection Check mit Venios angepasst auf neue Schnittstelle
- 05 Nov 2020 EsiConnectModuleService_Release_bb35fb80.zip[61]
  Changes:
    - Neue Schnittstelle Venios (neuer Login-Token + neue WebSockets-Schnittstelle)
    - Wichtig: im appsettings.json müssen folgende Parameter umgestellt werden:
      "VeniosClientUrl": "https://remap_newauth.venios.de[62]:8282/",
      "VeniosWebSocketUrl": "wss://remap_newauth.venios.de:8282/api/getwebsocketSSL?username={0}&password={1}",
- 19 Oct 2020 17:30 EsiConnectModuleService_Release_f841cfd3.zip[63]
  Changes:
    - Bugfix (2. Versuch): Fehlende Verbindung zu Venios wird nicht korrekt erkannt (nach Wegnahme von Internet-Verbindung ist der ReMaP-Watchdog weiter gelaufen)
    - Umstellung für "PeriodicOnChange"-Signale: Samplingrate wird jetzt für das SampleInterval auf dem MonitoredItem verwendet (statt als PublishInterval auf der Subscription)
      → Hier wäre interessant zu sehen, ob es immer noch Lücken gibt, obwohl sich der Messwert verändert hat.
- 19 Oct 2020 14:30 EsiConnectModuleService_Release_2d987fa8.zip[64]
  Changes:
    - Bugfix: Fehlende Verbindung zu Venios wird nicht korrekt erkannt (nach Wegnahme von Internet-Verbindung ist der ReMaP-Watchdog weiter gelaufen)
    - Erweiterung: Weniger Subscriptions - Es wird jetzt pro Publishing Interval nur noch eine Subscription erzeugt. So lange nicht mehr als 9 verschiedene SamplingRates für die Timeline-Signale konfiguriert werden, sollte es funktionieren.

---

59 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_bcf40f6a.zip?api=v2&modificationDate=1616422342855&version=1
60 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_19e05ff1.zip?api=v2&modificationDate=1605101013442&version=1
61 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_bb35fb80.zip?api=v2&modificationDate=1604568930288&version=1
62 http://venios.de
63 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_f841cfd3.zip?api=v2&modificationDate=1603121939787&version=1
64 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_2d987fa8.zip?api=v2&modificationDate=1603111187670&version=1

- 15 Oct 2020 EsiConnectModuleService_Release_b382fe13.zip[65]
  Changes:
  - Bugfix: Signale welche OnChange gespeichert werden, werden nicht gespeichert, wenn beim Starten des ConnectModules kein Lese-Zugriff auf dem OPC UA-Server gegeben ist.
- 07 Oct 2020 EsiConnectModuleService_Release_2030d3e8.zip[66]
  Changes:
  - OPC UA-Subscriptions werden beim Stoppen des ConnectModules aufgehoben
  - Bugfix: ConnectModul kann auch ohne konfigurierten Control-Signale betrieben werden
- 28 Sep 2020 EsiConnectModuleService_Release_00b8dfcd.zip[67]
  Changes:
  - Proof-Of-Concept-Implementierung von: Periodic/onChange-Sampling für periodisch konfigurierte Signale
    Signal is recorded periodically every 5min (the sampling rate is the same for all periodic signals of one ConnectModule)
    Additionally the signal is recorded onChange, but not faster than the provided sampling rate from the signallist.
  - Bugfix: auch nach längeren (Verbindungs-)Problemen wird weiterhin zeitnah aufgezeichnet
  - Verbessertes Logging
- 31 Aug 2020 EsiConnectModuleService_Release_069e3f5b.zip[68]
  Changes:
  - Kein Überschreiben der Config beim Update
  - Vollständiges Disable des Services in StopAndDisable-Skript
  - 26 Aug 2020 EsiConnectModuleService_Release_b7aa96a8.zip[69]
    Changes:
    - Besseres Logging an Schnittstelle zu Venios und OPC UA
  - EsiConnectModuleService_Release_ecfe5db5.zip[70]

3.2.1.2 SignallistTranslator

Newest package:

- 19 Mar 2021 EsiSignalListTranslator_Release_bcf40f6a.zip[71]

History:

- 12 Oct 2020 EsiSignalListTranslator_Release_afc20683.zip[72]
  Changes:
  - Error-Signale zu jedem Lese-Signal (Timeline und Status) werden automatisch hinzugefügt
- 13 Jul 2020 esi-SignallistTranslator.zip[73]

---

65 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_b382fe13.zip?api=v2&modificationDate=1602772910362&version=1
66 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_2030d3e8.zip?api=v2&modificationDate=1602256211125&version=1
67 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_00b8dfcd.zip?api=v2&modificationDate=1602256217956&version=1
68 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_069e3f5b.zip?api=v2&modificationDate=1602256227577&version=1
69 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_b7aa96a8.zip?api=v2&modificationDate=1602256237485&version=1
70 https://tools.scs.ch/wiki/download/attachments/110266319/EsiConnectModuleService_Release_ecfe5db5.zip?api=v2&modificationDate=1602256247053&version=1
71 https://tools.scs.ch/wiki/download/attachments/110266319/EsiSignalListTranslator_Release_bcf40f6a.zip?api=v2&modificationDate=1616167424603&version=1
72 https://tools.scs.ch/wiki/download/attachments/110266319/EsiSignalListTranslator_Release_afc20683.zip?api=v2&modificationDate=1602489396043&version=1
73 https://tools.scs.ch/wiki/download/attachments/110266319/esi-SignallistTranslator.zip?api=v2&modificationDate=1602256140094&version=1

Verwendung: (siehe auch Integration ESI - Oct 2020[74])

- Im Skript *Run ESI-SignallistTranslator.cmd* sieht man wie der SignallistTranslator verwendet wird.
- PSI kann ein eigenes CSV erstellen und entsprechend das Skript anpassen.
- Fehler werden auf der Commando-Zeile und im Errors.txt-File angezeigt. Weitere Details finden sich im log (im Ordner "log")
- Das File EsiSystemConfigFile.json ist der Output. Dies ist die Konfiguration für das ConnectModule.

3.2.2 Betrieb ESI-ConnectModule

3.2.2.1 Current package

Siehe: Releases for PSI(see page 61)

3.2.2.2 Configuration

The ESI-ConnectModule can be configured with the following files, all found under **C:\Program Files\ReMaP\EsiConnectModuleService**

Config-File	**Content**
appsettings.json	General settings for ConnectModule, like connection strings, polling intervals, ...
credentials.json	credentials to other systems
EsiSystemConfigFile.json (generated by SignalListTranslator)	ReMaP-specific configuration: which signals are archived, which signals can be controlled. This file corresponds to the data from the signal list (it is the output of the SignallistTranslator).

74 https://tools.scs.ch/wiki/display/SGSREMAP/Integration+ESI+-+Oct+2020

Config-File	Content
Opc.Ua.esiConnectModule.Config.xml	Configuration for the OPC UA-interface to ESI. The following fields are configured in other files: ▪ URL: appsettings.json ▪ Credentials: credentials.json

3.2.2.3 Log

To analyse problems or to check if the ConnectModule runs properly, check its newest log-file, under **C:\Program Files\ReMaP\EsiConnectModuleService\log**

3.2.2.4 Install & Update

Install

When you first install the Connect Module the following steps need to be done:

1. Install
2. Start
3. Copy Certificates for OPC UA-Connection
4. Stop
5. Start

Update

Once a first installation is in place, the update procedure is:

1. Stop
2. Update
3. Start

Exchange Configuration

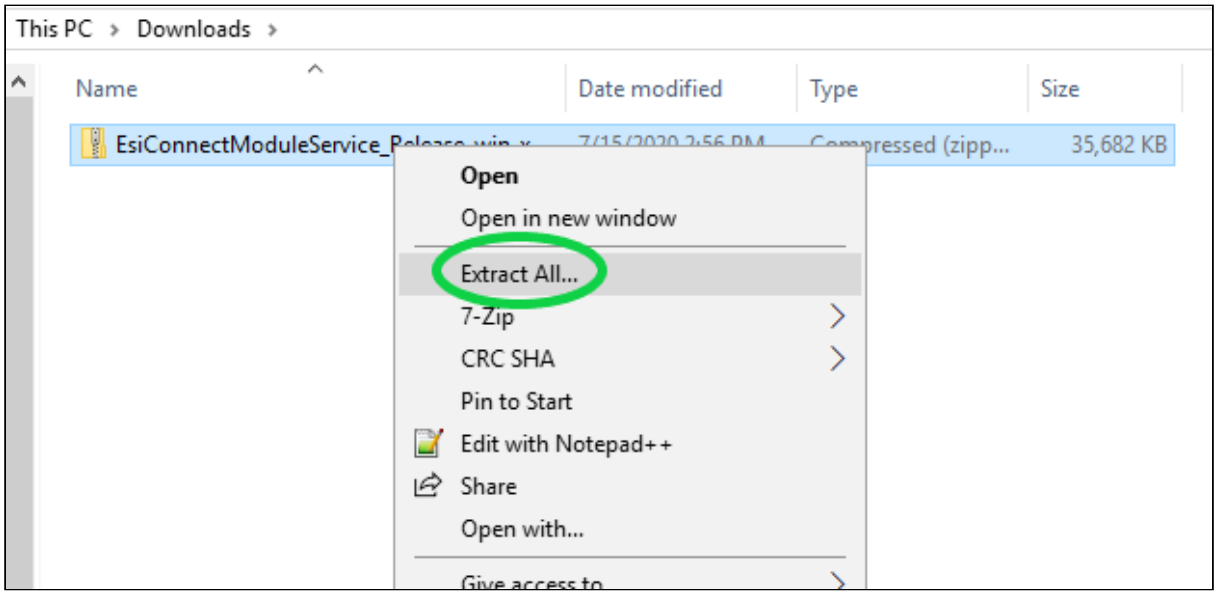
When you make changes to one of the Configuration files, the procedure is:

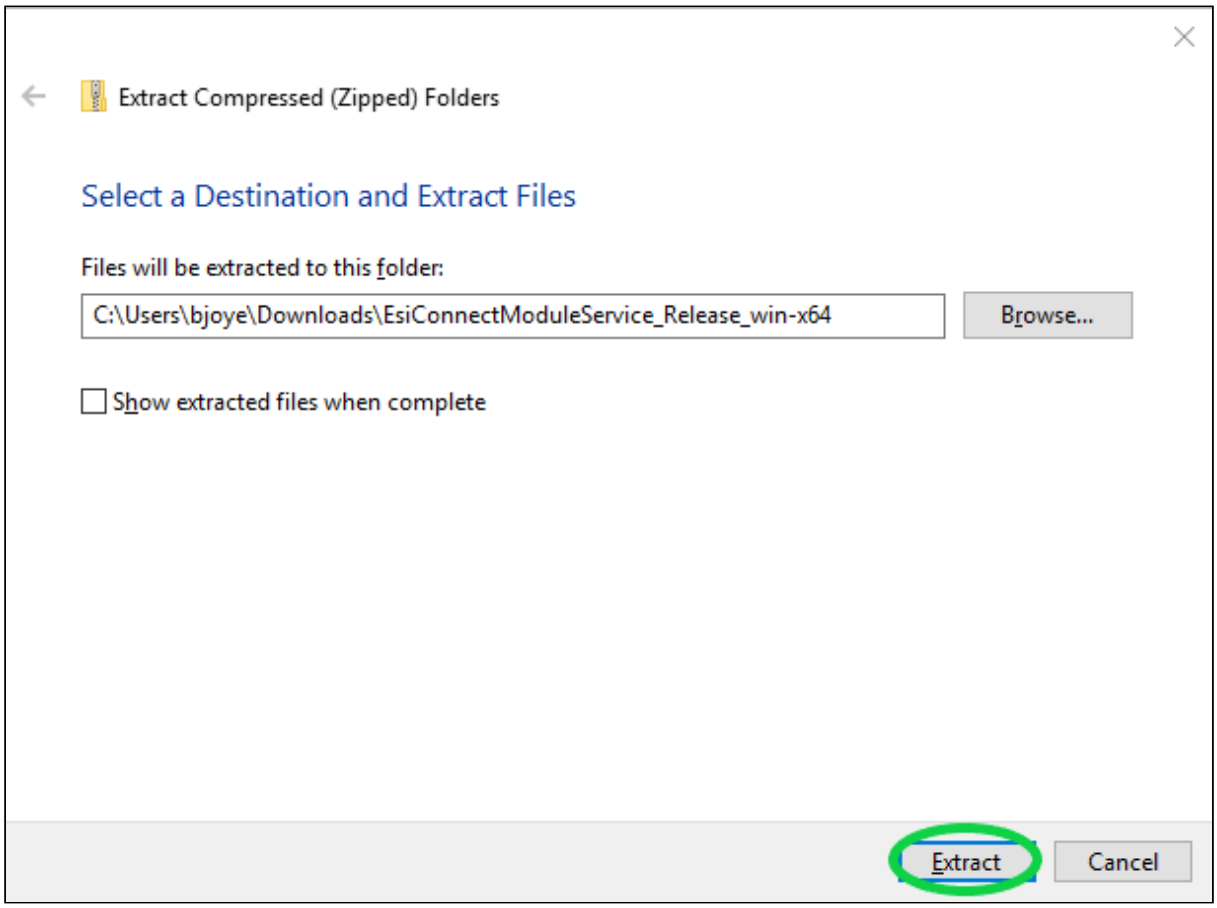
1. Replace Configuration file or edit it with administrator rights
2. Stop
3. Start

Install

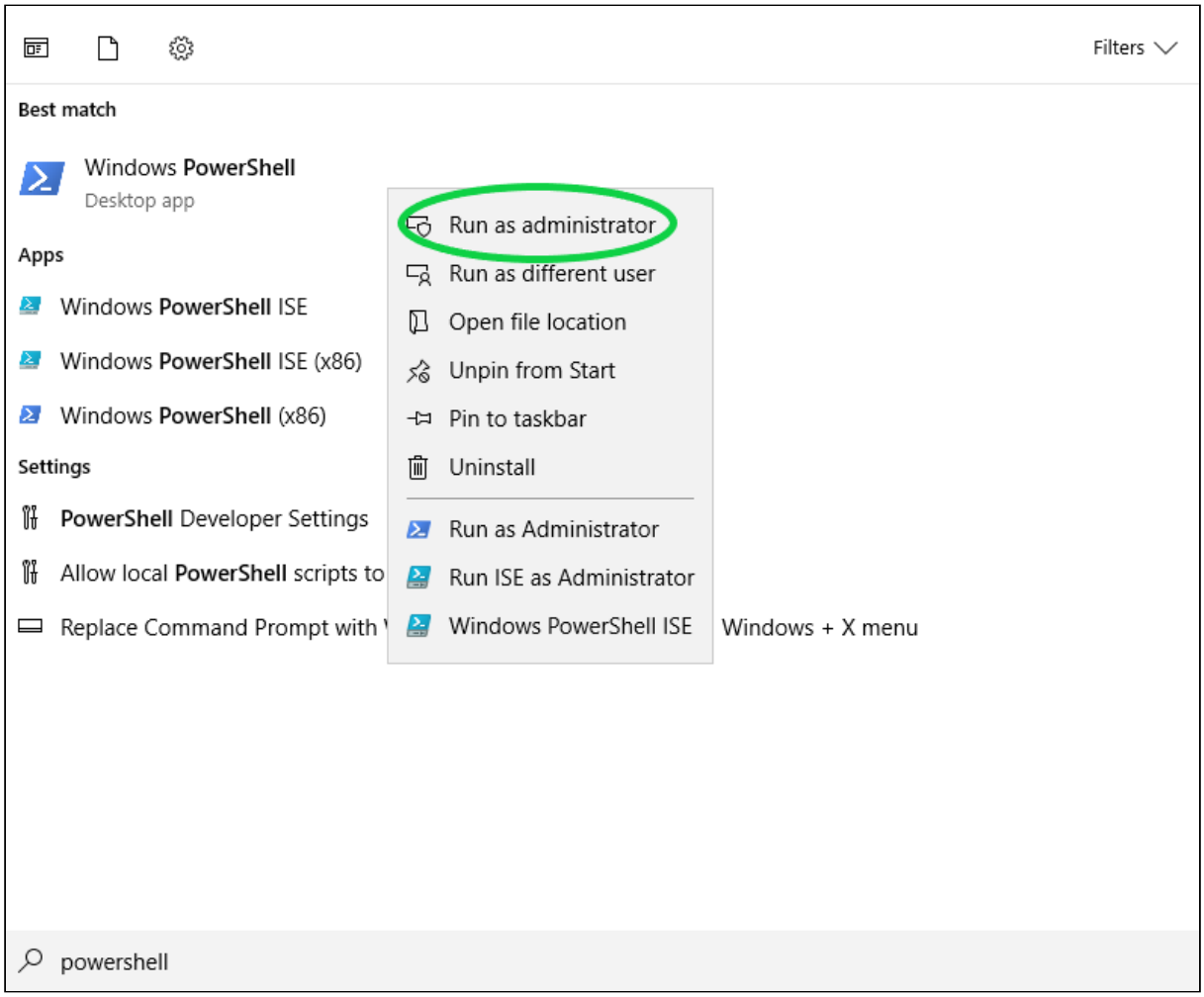
You should receive a `.zip` file from SCS which contains the newest version of the Esi Connect Module as a Windows Service. Follow these instructions to install the Connect Module on your Windows machine:

1. Extract all the contents of the `.zip` file to a temporary directory (e.g., Downloads). It does not matter where you extract the contents.
   This can be done easily by right-clicking the file and selecting "Extract All...":



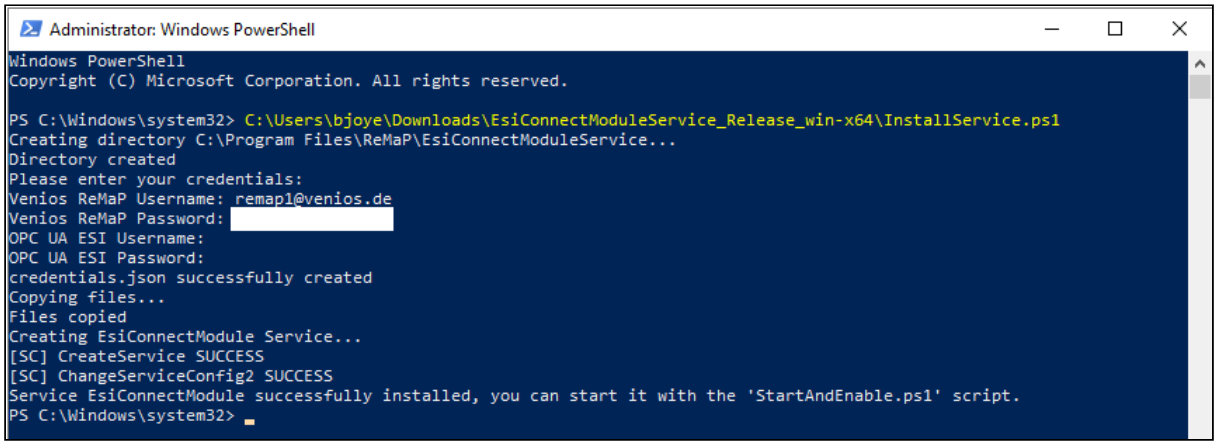


2. Search the keyword "PowerShell" with the Windows Search, and right-click the `Windows PowerShell` result. Finally, select "Run as administrator":



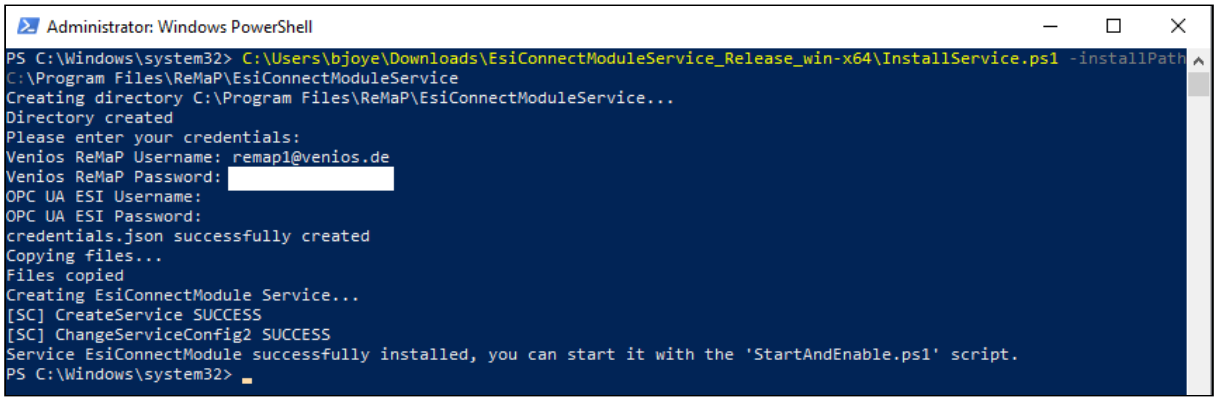
Note: You need administrator rights on the machine in order to install Windows Services.

3. Call the `InstallService.ps1` PowerShell script by typing its absolute path. You will be prompted to enter your credentials

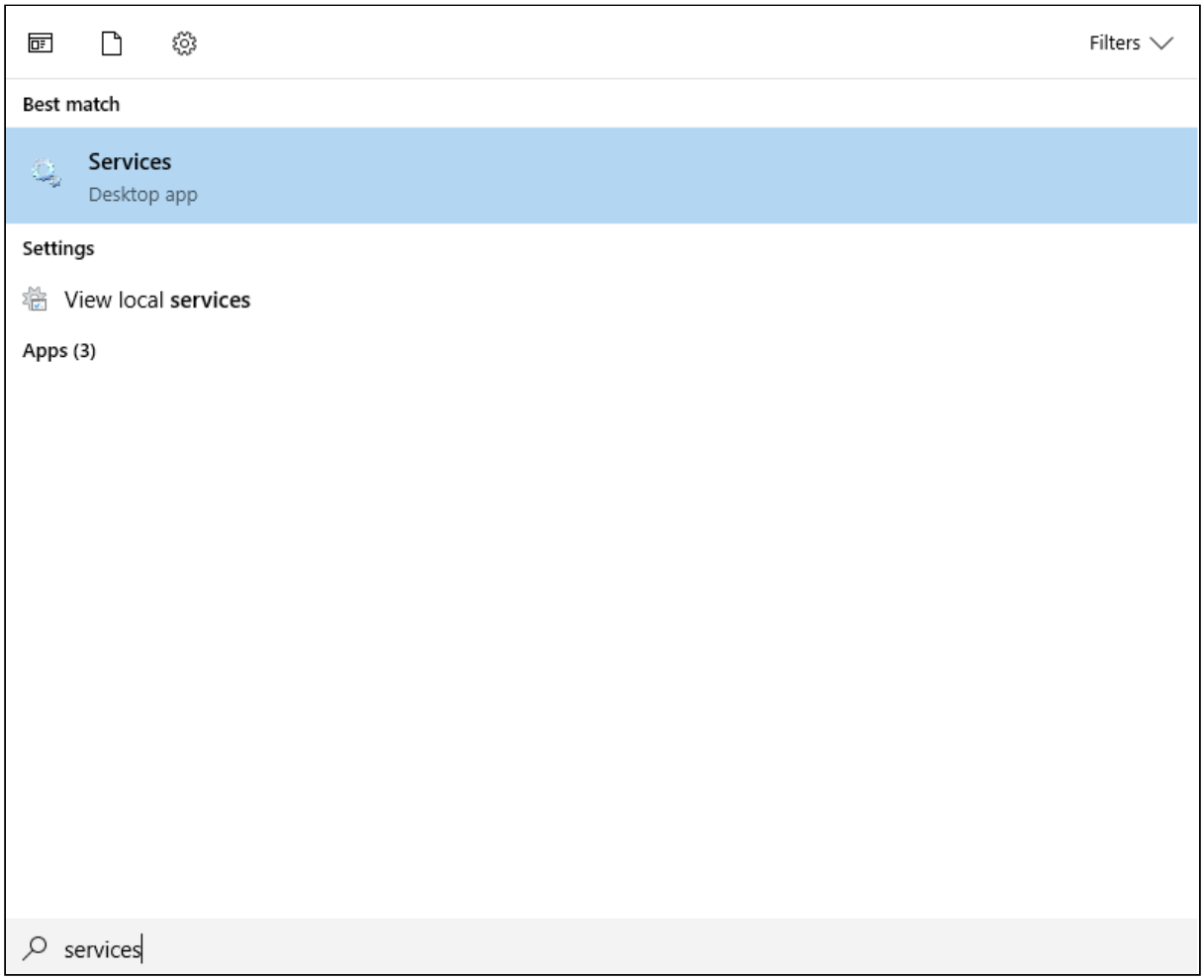


If everything goes right, you should see `[SC] CreateService SUCCESS` .

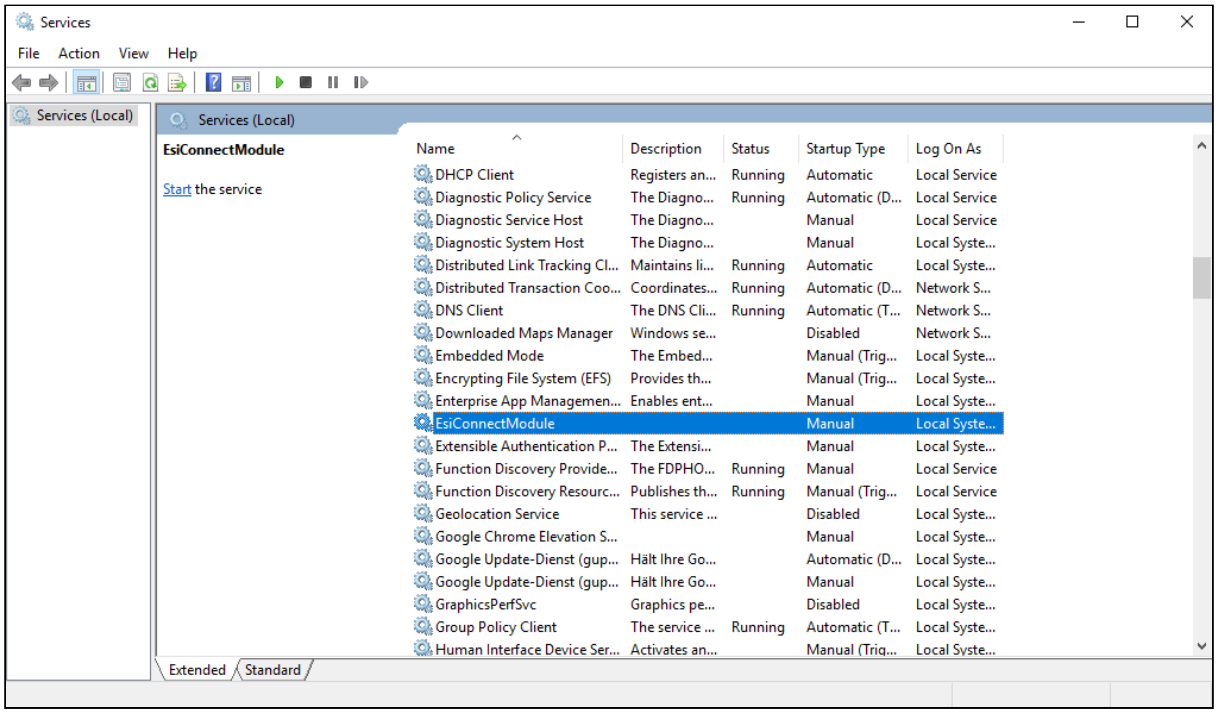
Note: The default install location is `C:\Program Files\ReMaP\EsiConnectModuleService` , you can however choose your own path by setting the parameter `installPath` . Simply append `-installPath your\own\path` to the original command:



4. Check that the EsiConnectModule Service was added by searching and opening the keyword "Services" with the Windows Search:



You might need to scroll down to your Service:



Congratulations, you successfully installed the Connect Module as a Windows Service!

Start

You can start the EsiConnectModule Service by calling the `StartAndEnable.ps1` PowerShell script (typing its absolute path in an admin PowerShell):

```
PS C:\Windows\system32> C:\Users\bjoye\Downloads\EsiConnectModuleService_Release_win-x64\StartAndEnable.ps1
[SC] ChangeServiceConfig SUCCESS
Service EsiConnectModule successfully started
PS C:\Windows\system32>
```

Once started, the Connect Module will create a log file in a new `log` folder in your installation path and record its outputs in the installed location. Per default, you will find the log files in `C:\Program Files\ReMaP\EsiConnectModuleService\log`.

Behaviour

The "Enable" part of the script makes sure that the service will be started automatically on reboot.

Additionally, the service will auto restart on failure, following this pattern: restart after 2 seconds on first failure, after 30 seconds on second failure, after 5 minutes on third failure, and finally it will try to restart once per hour on each subsequent failure. The failure count is reset to 0 when the service runs for 5 consecutive minutes without failures.

Stop

You can stop the EsiConnectModule Service by calling the `StopAndDisable.ps1` PowerShell script (typing its absolute path in an admin PowerShell):

```
PS C:\Windows\system32> C:\Users\bjoye\Downloads\EsiConnectModuleService_Release_win-x64\StopAndDisable.ps1
[SC] ChangeServiceConfig SUCCESS
Service EsiConnectModule successfully stopped
```

Behaviour

The "Disable" part of the scripts makes sure that the service will **not** be started automatically on reboot.

Update

To update an already existing EsiConnectModuleService, you first need to stop the EsiConnectModule Service, and then call the `UpdateService.ps1` script:

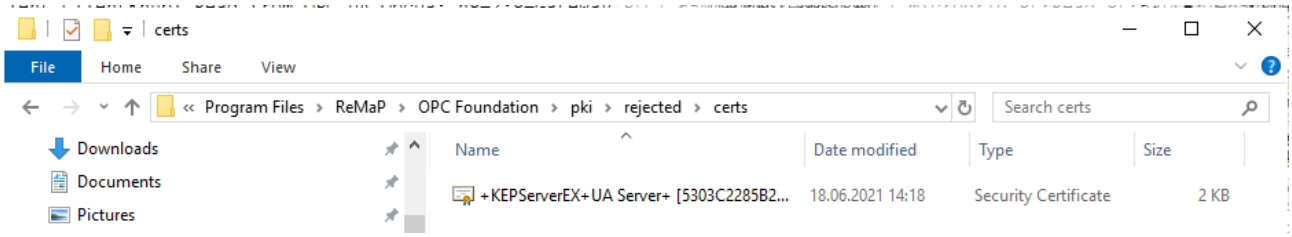
1. Start PowerShell as administrator (See point 2. of Installing)
2. Call the `StopAndDisable.ps1` PowerShell script by typing its absolute path.
3. Call the `UpdateService.ps1` PowerShell script by typing its absolute path:

```
PS C:\Windows\system32> C:\Users\bjoye\Downloads\EsiConnectModuleService_Release_win-x64\StopAndDisable.ps1
[SC] ChangeServiceConfig SUCCESS
Service EsiConnectModule successfully stopped
PS C:\Windows\system32> C:\Users\bjoye\Downloads\EsiConnectModuleService_Release_win-x64\UpdateService.ps1
Copying files...
Files copied
EsiConnectModule service successfully updated, you can start it with the 'StartAndEnable.ps1' script.
PS C:\Windows\system32>
```

Copy Certificates for OPC UA-Connection

The certificate for the OPC UA connection must be stored in the correct "trusted"-folder. After the ConnectModule is executed for the first time it is automatically stored in the "rejected"-folder. Therefore:

Move certificate from rejected folder **C:\Program Files\ReMaP\OPC Foundation\pki\rejected\certs**



to trusted-folder **C:\Program Files\ReMaP\OPC Foundation\pki\trusted\certs**

These folders are configured in the Opc.Ua.esiConnectModule.Config.xml-file (as TrustedPeerCertificates resp. RejectedCertificateStore).

3.2.3 ESI Signallist Translator

3.2.3.1 Aktuelle Version

siehe Releases for PSI(see page 61)

3.2.3.2 Workflow

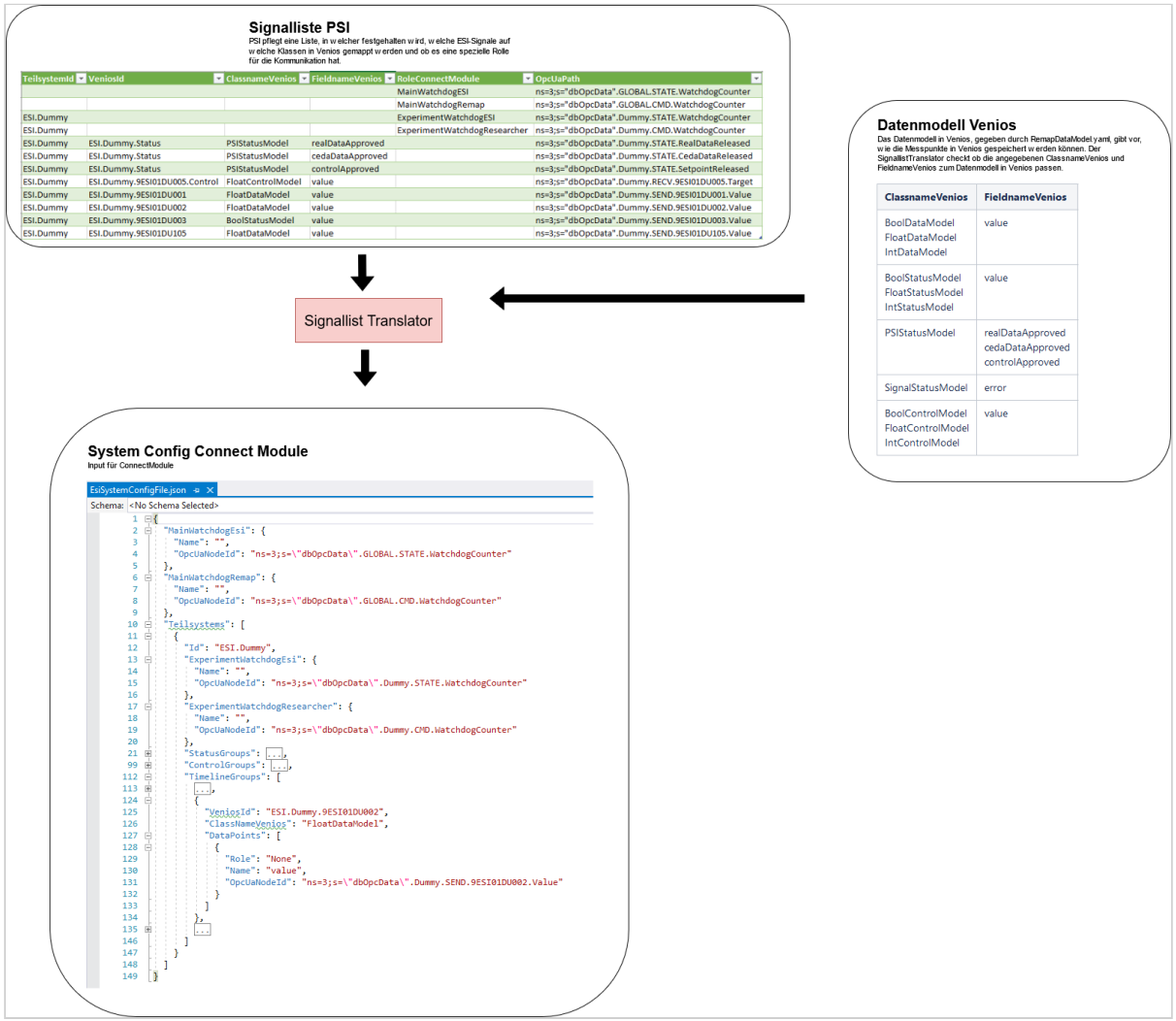
Der Signallist Translator generiert das EsiSystemConfigFile.json für das ConnectModule.

Input files:

- Signalliste_esiDummy.csv[75]
- RemapDataModel.yaml[76] (dies entspricht https://remap.venios.de:8282/swagger/v4/swagger.json)

Output file:

- EsiSystemConfigFile.json[77]



75 https://tools.scs.ch/wiki/download/attachments/126527499/Signalliste_esiDummy.csv?api=v2&modificationDate=1629893272980&version=2
76 https://tools.scs.ch/wiki/download/attachments/126527499/RemapDataModel.yaml?api=v2&modificationDate=1629891941671&version=1
77 https://tools.scs.ch/wiki/download/attachments/126527499/EsiSystemConfigFile.json?api=v2&modificationDate=1629893293095&version=2

3.2.3.3 Constraints

Column	Description	Restrictions

TeilsystemId	Id of the Teilsystem the signal belongs to (e. g. ESI.ELY, ESI.Cosyma, ...)	• The following characters are allowed:
		  • letters: A-Z, a-z
		  • digits: 0-9
		  • special characters: _ .
		• The first character shouldn't be a digit
		• Can be empty

VeniosId	Identifier that is used in Venios.
	The triplet (VeniosId, ClassnameVenios, FieldnameVenios) identifies how the data can be accessed in Venios.
		▪ The triplet VeniosId, ClassnameVenios, FieldnameVenios must be unique when defined.
		▪ The following characters are allowed:
		  • letters: A-Z, a-z
		  • digits: 0-9
		  • special characters: _ .
		▪ The first character shouldn't be a digit
		▪ Can be empty

<table>
<tr><th>Column</th><th>Description</th><th>Restrictions</th></tr>
<tr><td>ClassnameVenios

FieldnameVenios</td><td>Defines the Class and fieldname in Venios, that is used to persist the signal. Classname, Fieldname and VeniosId identify how the data can be accessed in Venios.

The naming of the ClassnameVenios also specifies, how the data is stored:
(...)DataModel
Signal is persisted periodically and when it changes its state
(...)StatusModel
Signal is persisted periodically and when it changes its state
(...)ControlModel
Signal represents a control-value, that can be set over the Venios-REST-API. Signal is persisted periodically and whenever it changes its state.
empty
The signal will not be persisted in Venios.</td><td>If set, then it has to be one of the following combinations of ClassnameVenios and FieldnameVenios (defined in the RemapDataModel.yaml):<table><tr><th>Classname Venios</th><th>Fieldname Venios</th><th>Usage</th></tr><tr><td>BoolDataModel
FloatDataModel
IntDataModel</td><td>value</td><td>Measurement data</td></tr><tr><td>BoolStatusModel
FloatStatusModel
IntStatusModel</td><td>value</td><td>Status data</td></tr><tr><td>PSIStatusModel</td><td>realDataApproved
cedaDataApproved
controlApproved</td><td>Status of Control-Device (needed to check control access)</td></tr><tr><td>SignalStatusModel</td><td>error</td><td>Error data</td></tr><tr><td>BoolControlModel
FloatControlModel
IntControlModel</td><td>value</td><td>Signal represents a control-value, that can be set over the Venios-REST-API.</td></tr></table>
combination VeniosId, ClassnameVenios, FieldnameVenios must be unique.</td></tr>
</table>

Column	Description	Restrictions
RoleConnectModule	Field with specific logic in the ConnectModule: • (empty) or None: field has no role • MainWatchdogEsi = Watchdog that is written by ESI. ESI expects the same value to come back with a signal with the role MainWatchdogRemap. • MainWatchdogRemap = Watchdog that is written by the ConnectModule. This signal must reflect the signal with the role MainWatchdogEsi back to ESI. The MainWatchdog(...) roles allow to check the connection between Venios and ESI. • ExperimentWatchdogEsi = Watchdog that is written by ESI. The ConnectModule receives the signal. • ExperimentWatchdogResearcher = Watchdog that is written by the ConnectModule, to communicate if the researcher is still connected. The ConnectModule responds only if it knows that the researcher is still connected. The ExperimentWatchdog(...) roles allow to check the connection between the researcher and ESI.	▪ For the **whole system** (ESI), there must be ▪ exactly one signal pair (MainWatchogEsi, MainWatchogRemap). ▪ A **Teilsystem**, that has at least one signal with ▪ ClassnameVenios (...)ControlModel, must have ▪ exactly one signal pair (ExperimentWatchdogEsi, ExperimentWatchdogResearcher)
OpcUaPath	Full OPC UA path to the signal. This is the same as the NodeId displayed in UaExpert	Not empty
~~Samplerate~~	~~Sample interval in seconds. Data is periodically persisted at this rate.~~ ~~(Remark: may be changed to SampleInterval in the future, that is the correct term)~~	Not in use anymore - value will be ignored. Samplerate is configured for all signals globally in appsettings.json of ConnectModule.

Fileformat

- Die Signalliste ESI ist ein File im Format CSV gemäss RFC 4180[78].
- Als Zeichen für die Trennung von Datenfeldern wird das Komma (,) verwendet.
- Sollte ein Feld ein Komma enthalten (was eigentlich nicht vorkommen sollte), muss das Feld mit Anführungszeichen (") umschlossen sein.
- Die erste Zeile muss eine Kopfzeile mit den Spaltenbezeichnern sein.

78 https://tools.ietf.org/html/rfc4180

Das Tool "Signallist Translator" überprüft Restrictions aus obiger Tabelle. Das Tool "Signallist Translator" erzeugt seinen Output nur, wenn es keinen Fehler in der Signalliste ESI findet. Ein Run des Tool wird nicht beim ersten Fehler diesen ausgeben und stoppen, sondern analog wie ein Compiler immer alle Fehler melden.

3.2.3.4 Watchdogs

Description

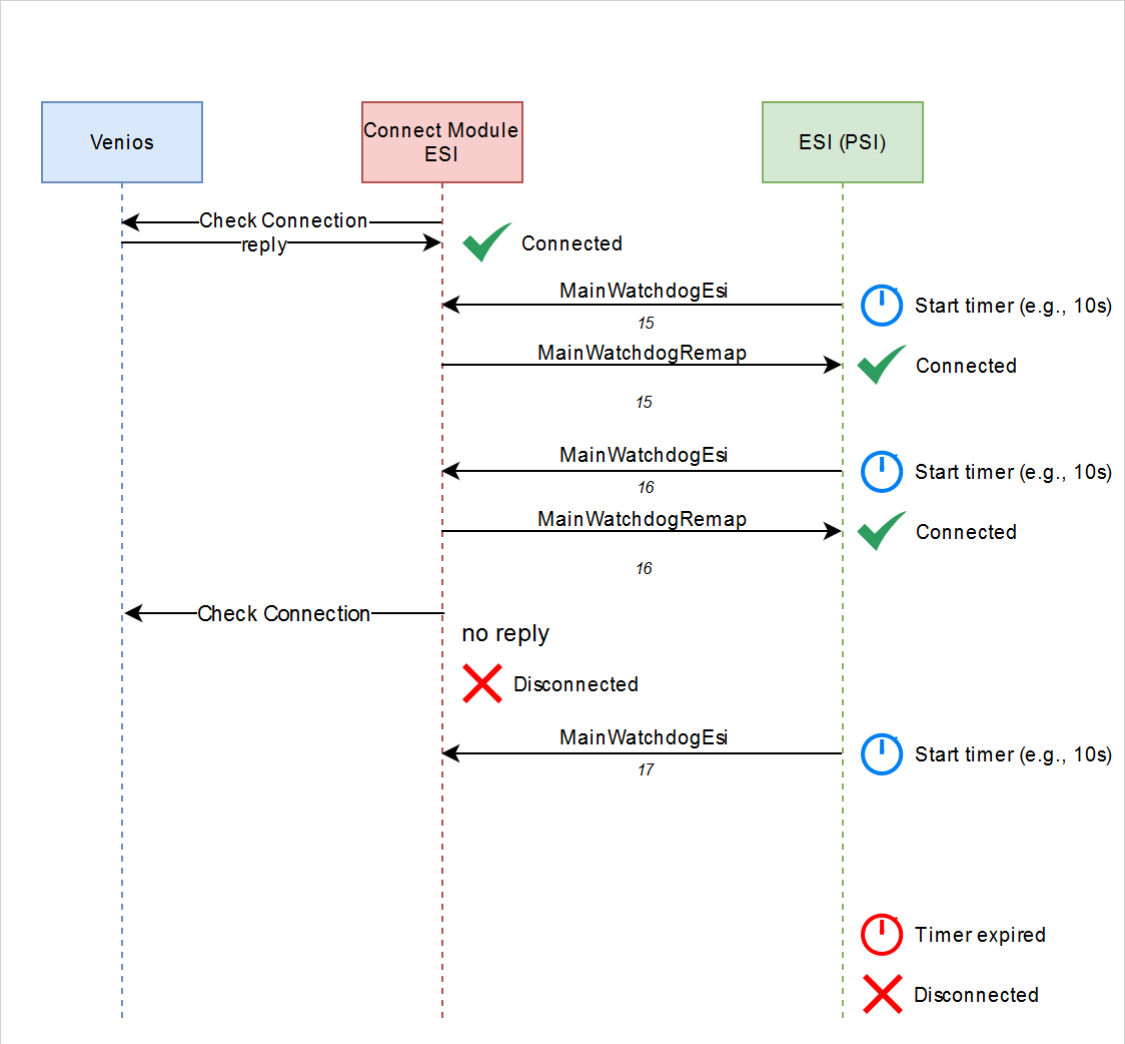
A watchdog is always defined by a pair of crossing signals. There are two types of watchdogs:

- Main watchdog: establishes the connection between ESI and Remap. There is only one main watchdog for the whole system.
- Experiment watchdog: establishes the connection between ESI and the researcher. The signal is not actually sent to the researcher. The ConnectModule handles the watchdog while listening if the researcher is still online. There is one experiment watchdog for each Teilsystem where an experiment is running.

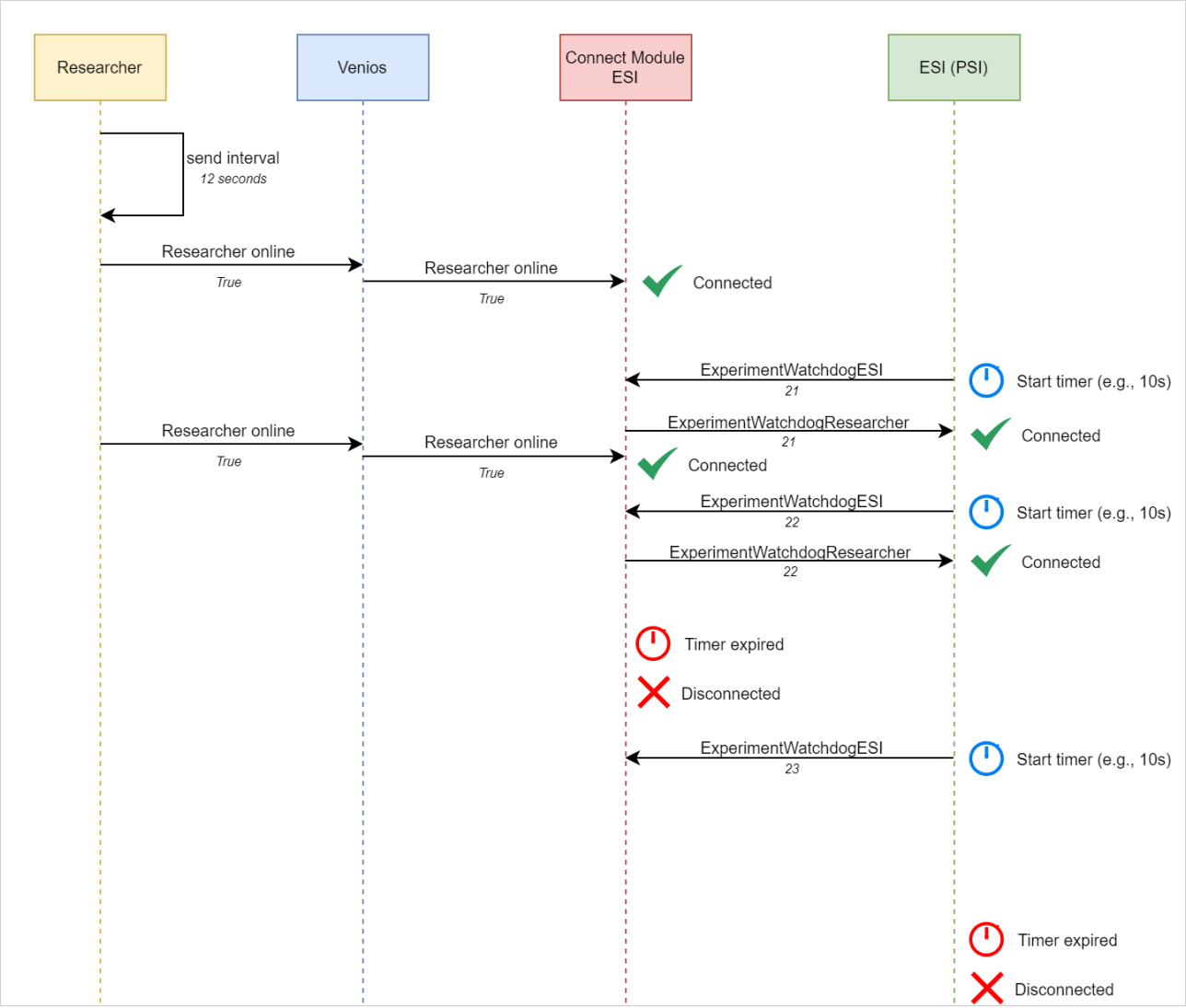
Mechanism

The watchdog signal is an integer. It is sent by ESI to the ConnectModule (main watchdog) or to the researcher (experiment watchdogs) represented by the ConnectModule. The receiver of the watchdog must send the same integer back to ESI. A timeout timer is started at ESI when the signal is sent. If ESI doesn't get the same integer back in time, it means that there is a connection problem between ESI and Remap. If the correct signal was received, ESI reiterates the operation with a different integer (typically by incrementing the integer).

The example of the main watchdog is shown in the following diagram:



The example of an experiment watchdog is shown in the following diagram:



3.2.3.5 CEDA/real Daten Freigabe

Es gibt 2 verschiedene Datentypen "real" und "CEDA". Pro Teilsystem gibt es separate Freigaben für "real"-Daten und "CEDA"-Daten.

Umsetzung:

- ESI schickt zu jedem analog und digital Signal ein zusätzliches Bool-Flag "Access", welches true ist, sofern ReMaP das Signal lesen darf
- Der konfigurierte OPC UA Pfad enthält nicht einen einzelnen Datenpunkt, sondern ein ganzes Objekt, welches ein Feld "Value" (für den eigentlichen Wert) und ein Feld "Access" (für den Zugriff) beinhaltet. Sofern "Access" auf False steht, wird der "Value" ignoriert und nicht weiter verarbeitet (technische Anmerkung: dies wird direkt in der Client-Klasse gemacht).

3.2.3.6 Offene Punkte

Folgende Features wurden nicht umgesetzt:

- Enums/Status-Informationen:
  Idealerweise gäbe es zu Integer-Signalen, bei denen ein Wert eine bestimmte Bedeutung hat (z. B. 0 - on, 1 - starting, 2- running, 3 - stopping, 4 - error), die Möglichkeit diese Bedeutung im Signallisten-Excel zu hinterlegen und auf der Venios-Schnittstelle hinterlegt zu haben.
- Einheiten:
  Idealerweise könnte man bei der Kommunikation mit Venios eine Einheit mitgeben. Zum einen beim Speichern, aber auch beim Abfragen von gespeicherten Werten. Für diesen Lösungsansatz, müssten die Einheiten im Signallisten-Excel hinterlegt werden.

3.3 Migration Venios V3.0 -> V4.0

To migrate to the new version V4.0 of Venios the following steps have to be taken:

- Migrate Signallists (resp. DeviceConfig/SystemConfig)
- Use new Version of SignallistTranslator

Deploy new Version of ConnectModule

3.3.1 Migration Signallists

The DeviceConfig (ehub) resp. SystemConfig (ESI) remain in the same format. But ClassnameVenios and FieldnameVenios have to be adjusted to the new names in Venios:

ALT (V3.0)		**NEU (V4.0)**	
ClassnameVenios	**FieldnameVenios**	**ClassnameVenios**	**FieldnameVenios**
BooleanTimelineDataModel FloatTimelineDataModel IntegerTimelineDataModel	Value	BoolDataModel FloatDataModel IntDataModel	value

ALT (V3.0)		NEU (V4.0)	
ClassnameVenios	**FieldnameVenios**	**ClassnameVenios**	**FieldnameVenios**
BooleanStatusModel FloatStatusModel IntegerStatusModel	Value	BoolStatusModel FloatStatusModel IntStatusModel	value

ALT (V3.0)		NEU (V4.0)	
ClassnameVenios	**FieldnameVenios**	**ClassnameVenios**	**FieldnameVenios**
EhubStatusModel	AlarmActiveResearchModeConfirmationResearchMod	EHubStatusModel	alarmActiveresearchModeConfirmationresearchModeApprovalalarmStatusresearchModeStatus

ALT (V3.0)		NEU (V4.0)	
ClassnameVenios	**FieldnameVenios**	**ClassnameVenios**	**FieldnameVenios**
	eApprovalAlarmStatusResearchModeStatus		

ALT (V3.0)		NEU (V4.0)	
ClassnameVenios	**FieldnameVenios**	**ClassnameVenios**	**FieldnameVenios**
EsiStatusModel	RealDataApprovedCedaDataApprovedControlApproved	PSISStatusModel	realDataApprovedcedaDataApprovedcontrolApproved

ALT (V3.0)		NEU (V4.0)	
ClassnameVenios	**FieldnameVenios**	**ClassnameVenios**	**FieldnameVenios**
SignalStatusModel	Error	SignalStatusModel	error
BooleanControlModel FloatControlModel IntegerControlModel	Value	BoolControlModel FloatControlModel IntControlModel	value

4 System Development

4.1 Anbindung neue Version Venios V4.0

Im folgenden sind die nötigen Anpassungen aufgelistet, welche durchgeführt werden müssen, um die neue Version Venios V4.0 anzubinden. Ferner ist beschrieben, welche Arbeiten für d

4.1.1 Dokumente

Link zu Swagger-Docu (produktiv): https://remap.venios.de:8282/index.html

Schnittstellen-Beschreibung: Vep4.0_Remap_v1.4.pdf[79]

4.1.2 Signallist Translator

⚠ Inputfile RemapDataModel.yaml[80] entspricht nicht mehr aktueller API

folgende Anpassungen müssen vorgenommen werden:

1. Datenmodell in RemapDataModel.yaml aktueller API anpassen
2. Signallistenfiles (*.csv) anpassen
   Spalte ClassnameVenios und ev. auch FieldnameVenios

Signallist-Translators sollte unverändert weiterverwendet werden können.

⚠ Ev. muss Logik implementiert/angepasst werden die Klassenname zu Status/Measurement/Control mappt.

✅ Done: RL 03.02.2021

4.1.3 Connect-Module

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status
Authentifizierung	neuer URL: POST /Apiv4/Authenticate/Login	ReMaP.Venios Client.Client				✅ Done RL: 03.02.2021

79 https://tools.scs.ch/wiki/download/attachments/117148019/Vep4.0_Remap_v1.4.pdf?api=v2&modificationDate=1611734907518&version=1
80 https://tools.scs.ch/wiki/download/attachments/86838546/RemapDataModel.yaml?api=v2&modificationDate=1589973084716&version=3

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status

vorhandene (Status/ Measurement/ Control)-Timelines prüfen und fehlende anlegen	neue URLs
	• GetAllTimelineVeniosIds: GET /Apiv4/Remap/ReadAll/MeasurementDevice
	  • ← Array MeasurementDeviceModel
	• CreateTimeline: POST /Apiv4/Remap/Create
	  • → MeasurementDeviceModel
	  • ← UUID
		• ReMaP.ehubConnectModule.ConnectModule.EhubConnectModule, ReMaP.esiConnectModule.ConnectModule.EsiConnectModule: muss alle Arten von Timelines prüfen
		• ReMaP.VeniosClient.Client
			• MeasurementDeviceModel.name = ReMaP.DataModel.Ehub.Device.VeniosId
			• ReMaP.DataModel.Ehub.SignalGroup.ClassName muss auf Enum in MeasurementDeviceModel gemappt werden
				• MeasurementDeviceModel.name = ReMaP.DataModel.Esi.SignalGroup.VeniosId
				• ReMaP.DataModel.Esi.SignalGroup.ClassName Venios muss auf Enum in MeasurementDeviceModel gemappt werden
					• Conncect-Module muss unter User laufen, der Recht zum Erzeugen von MeasurementDevice hat
					• Wie MeasurementDevice erzeugen mit genau einer Status-, Measurement- oder Control-Timeline?
					• Mit Venois im Detail klären was bei Create in MeasurementDeviceModel genau angegeben werden muss.
						✅ Done RL: 11.02.2021

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status
HistoricData entfallen		• Methode ReMaP.VeniosClient.IClient.AddToHistoric(..) entfernen • Klasse ReMaP.ConnectModuleCommon.Logger.HistoricLogger löschen und an deren Stelle ReMaP.ConnectModuleCommon.Logger.TimelineLogger verwenden				✅ Done RL: 11.02.2021 (Pull-Request noch offen)
Sample in Timeline speichern	neuer URL: POST /Apiv4/Remap/AddSingle/{MeasurementSourceUUID}/{TimelineType}	• ReMaP.ConnectModuleCommon.Logger.TimelineLogger.Translate(...) • ReMaP.VeniosClient.Client			Mit Venios im Detail klären was genau in Body geschickt werden muss.	✅ Done RL: 25.02.2021

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status
Auflösen MeasurementDevice Klarname → UUID	URLs: • GET /Apiv4/Remap/ComponentByName/{Name} • GET /Apiv4/Remap/ReadAll/MeasurementDevice Vermeiden dass bei jedem Speichern in Timeline zwei Calls nötig sind. Kann z.B. beim Programmstart Timeline prüfen/anlegen eine Map Klarname → UUID cachen.	ReMaP.ConnectModuleCommon.Logger.TimelineLogger muss Klarname → MeasurementSourceUUID übersetzen können	Klarname = ReMaP.DataModel.Ehub.Device.VeniosId	Klarname = ReMaP.DataModel.Esi.SignalGroup.VeniosId		✅ Done RL: 25.02.2021
Events senden	entfällt	Methode ReMaP.VeniosClient.IClient.FireEvent(..) entfernen (wird nur noch in Tests verwendet)				✅ Done RL: 11.02.2021

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status

Control-Wert schreiben	• Subscribe
	  • neuer URL: POST /Apiv4/Remap/Create
	  • → SubscriptionModel
	  • ← UUID
	• Unsubscribe
	  • neuer URL: DELETE /Apiv4/Remap/Delete/{Type}/{Uuid}
	
	ObservingSampler eines Control-Datenpunktes wird getriggert, wenn Datenpunkt in OPC-Server vom Connect-Module geschrieben/ verändert wird. Dies führt zum Speichern des Wertes zurück in die assoziierte Control-Timeline und Venios erzeugt wieder eine Event zum Schreiben des Control-Datenpunktes.
	
	Dieses Szenario kann wie folgt verhindert werden:
	• pro Control-Datenpunkt braucht es in Venios ein MeasurementDevice mit je einer Status- und einer Control-Timeline
	  • Control-Timeline
		• ReMaP.VeniosClient.IClient.Subscribe(???)
		• ReMaP.VeniosClient.Client
					• Mit Venois im Detail klären was bei Create in SubscriptionModel genau angegeben werden muss
					• bei Unsubscribe: Type = ?
						✅ Done RL: 26.02.2021

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status
	• Forscher/ Python-Code schreibt hier den zu schreibenden Wert • Connect-Module abonniert sich auf Measurement Device • Status-Timeline • Connect-Module schreibt hier den von ObservingSampler gelesen, aktuellen Control-Wert					

		betroffene Komponente	Ehub	Esi	Fragen/ Bemerkungen	Status
WebSocket für Subscription	separate WebSocket-Connection für jede Subscription: wss://[BaseURL]/apiv4/Stream/GetWebsocket/{Uuid}?username={usr}&password={pwd}				Mit Venois im Detail klären welche Daten genau via WebSocket empfangen werden. ⚠️ Ev. muss bei Reconnect nach Verbindungsproblem alte Subscription gelöscht und neue erzeugt werden.	✅ Done RL: 05.03.2021
Research-Mode triggern	Idee: Zum Triggern des Research-Mode (Ehub-Device) resp. Experiment-Watchdog (Esi-Teilsystem) wird ein spezielles MeasurementDevice mit einer Int-Control-Timeline erzeugt. Das Connect-Module abonniert auf dieses MeasurementDevice und Events stossen ResearchModeTrigger (Ehub) resp. ResearcherOnlineTrigger (Esi) an. Forscher/Python-Code muss regelmässig den Wert der Int-Control-Timeline schreiben (kann unverändert bleiben).		MeasurementDeviceModel.name = ReMaP.DataModel.Ehub.Device.VeniosId des Devices mit zwei Control-Datenpunkten in den Rollen ResearchModeWatchdog resp. ResearchModeRequest	MeasurementDeviceModel.name = ReMaP.DataModel.Esi.Teilsystem.Id Ist das eindeutig? Sollte sein.		✅ Done RL: 11.03.2021

4.1.4 Python-Code

- venios_api.py, checks.py anpassen/überarbeiten
  - Unterscheidung Timeline/Historic entfällt
  - neu: Auflösung Klarnamen → MeasurementDevice UUID
    z.B. read_signal.classname_venios → UUID
    soll bei Programmstart auflösen und z.B. in Signal-Objekt speichern (siehe checks)
  - anpassen: alle URL
- alle test_*****.py überarbeiten
  - tests "Historic Signal" entfallen
  - sollte sich auf anpassen Signal.classname_venios beschränken
- example_controller anpassen
  - ehub: muss laufen
  - esi: nur anpassen

4.1.5 Rechtemanagement für ein Experiment

Vorerst (Stand 30 Apr 2021) ist die neue Schnittstelle auf der Staging-Instanz aktiv: https://remap.venios.de:44334/index.html

prinzipieller Ablauf

1. Signallisten (*.csv) für Experiment erstellen/anpassen
2. mit Signallist-Translators Konfigurationen (*.json) für Connect-Module erstellen
3. ev. credentials.json anpassen
4. Connect-Module starten, damit in Venios fehlende Timelines (MeasurementDevices) angelegt werden
5. in Venios als "Administrator"
   a. neue Simulation erzeugen
      i. alle für Experiment benötigten MeasurementDevices hinzufügen (diese sollten vorhanden sein, da sie von Connect-Modulen ggf. erzeugt werden)
      ii. Zeitraum des Experiments festlegen
   b. neue UserGroup erzeugen
   c. die Simulation der UserGroup zuordnen
   d. ggf. einen neuen ReMaP-Nutzer (=Forscher) erzeugen
   e. ReMaP-Nutzer der UserGroup zuordnen

Idee zu Punkt 5: realisieren als Phyton-Skript + Config, wiederverwenden z.B. ***_signals.py mit Signalen eines Experimentes

weitere Use-Cases (Umsetzung aktuell nicht oder nur teilweise geplant):

- Zeitraum der Simulation verändern
- MeasurementDevices zu Simulation hinzufügen/entfernen
- ReMaP-Nutzer der UserGroup hinzufügen/entfernen
- aufräumen/löschen
  - Simulation
  - UserGroup
  - User
  - MeasurementDevice
  - allgemein: Löschen von Status/Measurement/Control-Daten eines Experimentes

4.1.6 Rechtemanagement (Doku Venios)

UserModel

Das UserModel ist die Repräsentation einer ActiveDirectory-Application die sich an der API einloggen kann. Es wird identifiziert an der Azure-ActiveDirectory Object Id. Aus technischen Gründen ist dies nicht die "uuid" des UserModels, sondern die "azureUserID". Es wird jedoch im Rest des Systems die "azureUserID" anstelle der "uuid" zur Identifizierung des User verwendet.

UserModels können nicht direkt angelegt werden, da sie fest mit den ActiveDirectory Applications verbunden sind. Ein UserModel wird automatisch angelegt, wenn ein neuer User sich anmeldet und zum ersten Mal eine Funktion ausführt, z.B. einen der Demo-API-Funktionen.

Ein UserModel kann, wenn es existiert, zu UserGroups zugeordnet werden.

Der uuid-Satz "userGroups" ist eine neue Property im UserModel um die Zuordnung verwalten zu können.

UserGroupModel

Das UserGroupModel fasst mehrere UserModels zusammen und kann Experimenten zugeordnet werden, um den Usern der UserGroup Lese-Berechtigung für die Daten des Experimentes, sowie Kontrollwert-Schreibrechte für die Dauer des Experiments zu geben. Die Zuordnung ist jeweils beidseitig möglich.
Für eine User-UserGroup Beziehung könnte also entweder beim UserModel in die UUID-Liste "userGroups" die Id der UserGroup eingetragen werden und dann über /Update geschrieben werden, oder bei der UserGroup wird die AzureID des users in die UUID-Liste "users" eingetragen und geupdatet. In beiden Fällen wird das jeweils andere Element automatisch mit geupdatet.

UserType

Ein UserModel kann einen von drei Typen haben:

- Administrator
- RemapUser
- ConnectModule

UserModels, UserGroups und Experimente können nur von einem Administrator geschrieben werden. Ebenfalls das Erstellen von Messgeräten und das Zuordnen von Messgeräten zu Experimenten ist Administratoren vorbehalten. Der ConnectModule UserType ist für den Moment deckungsgleich mit dem Administratoren-Typ.

Schreiben von Messwerten

Das Schreiben von Messwerten ist standardmäßig nur für den User möglich, der das Messgerät erstellt hat. Falls nötig kann hier eine API-Funktion angeboten werden, um anderen Usern oder UserGroups die Rechte für das Schreiben von Messwerten für ein bestimmtes Devices zu geben. Diese Schreibrechte können auf einen bestimmten Zeitraum eingeschränkt werden. Allerdings ist es hier wichtig zu beachten, dass sich der Zeitraum hier nicht auf den Zeitraum bezieht, für den Messwerte geschrieben werden können, sondern für den Zeitraum in dem Messwerte für beliebige Zeitstempel geschrieben werden können.

Lesen von Messwerten

Das Lesen von Messwerten ist für einen RemapUser nur über die API-Route die das Experiment enthält möglich. Außerdem können über diese Route jeweils nur Messwerte gelesen werden, dessen Zeitstempel zwischen dem Experiment-Start und Ende liegen.
Für einen Administrator können alle Messgeräte auch direkt ausgelesen werden, für beliebige Zeiträume.

4.2 ConnectModules

4.2.1 Connect Modules - Developer Setup

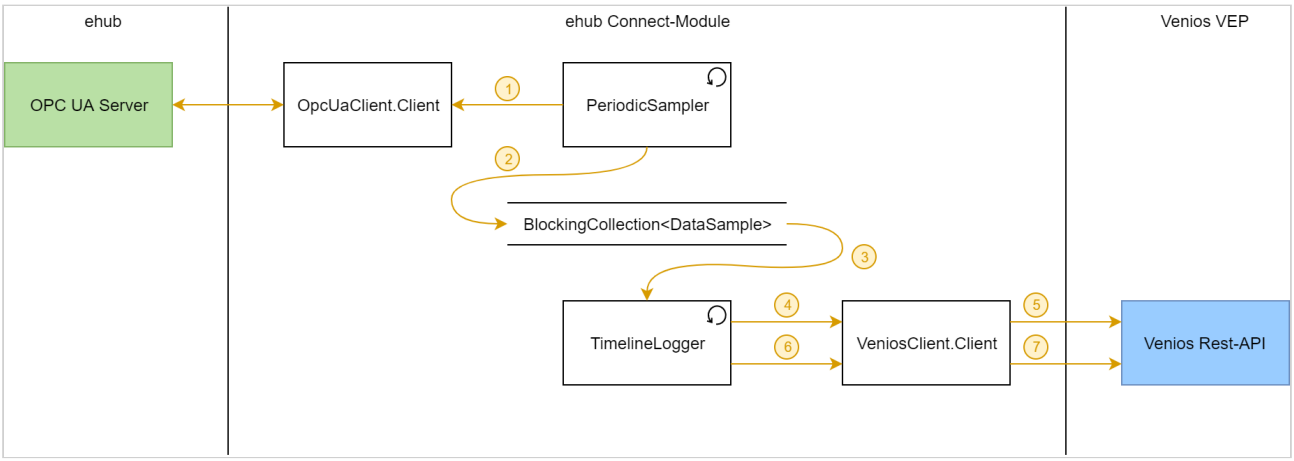
1. Git clone (currently only SCS has access):
   https://tools.scs.ch/code/projects/SGSREMAP/repos/venios_remap_connection
2. Execute script:
   ConnectModules\CreateCredentialsFile.ps1
   and enter credentials, which can be found in
   P:\Smart Grid Solutions\366-0_Allgemeines\ReMaP\Zugang\KeePassReMaP.kdbx
   credentials for the KeePass are given by Sabine Proll.
   The credentials are only accessible for SCS and should not be given out.
3. Open Solution ConnectModules\RemapConnectModules.sln in Visual Studio or Rider

4.2.2 ehub ConnectModule: Software-Design

4.2.2.1 Aufzeichnen Datenpunkte

Workflow: Aufzeichnen Messwerte

1. PeriodicSampler liest periodisch die Messwert-Datenpunkte eines Gerätes, packt diese in ein DataSample ...
2. ... und fügt das DataSample in die FIFO-Queue BlockingCollection<DataSample>.
3. TimelineLogger, getriggert durch das Einfügen des DataSample, empfängt das DataSample, übersetzt dieses in die Datenstruktur passed für Venios-Rest-API und ...
4. ... speichert die Daten mit VeniosClient.Client.AppendToTimeline(...).
5. VeniosClient.Client sendet Request apiV3/DataTimeline/Append
6. TimelineLogger meldet Event NewDataEvent mit VeniosClient.Client.FireEvent(...).
7. VeniosClient.Client sendet Request apiV3/Events/add



Workflow: Aufzeichnen weiterer Datenpunkte

Für die Aufzeichnung der Status-Datenpunkte, resp. der Kontroll-Datanpunkte eines Gerätes existieren analoge Datanflüsse, wobei

- anstelle eines PeriodicSampler abonniert sich ein ObservingSampler beim OpcUaClient.Client auf Veränderungen der Datenpunkte
- anstelle eines TimelineLogger übersetzt und speichert ein HistoricLogger die DataSample
- bei 4. werden die Daten mit VeniosClient.Client.AddToHistoric(...) gespeichert
- bei 5. wird Request apiV3/Historic/AddMany gesendet
- bei 6. wird Event StatusChangedEvent gemeldet

Übersicht

- PeriodicSampler: Erzeugt periodisch ein DataSample
- ObservingSampler: Erzeugt ein Sample, wenn sich ein zu beobachtender Datenpunkt ändert
- TimelineLogger: speichert Sample in Venios in Timeline-Daten
- Historic-Logger: speichert Sample in Venios in Historic-Daten

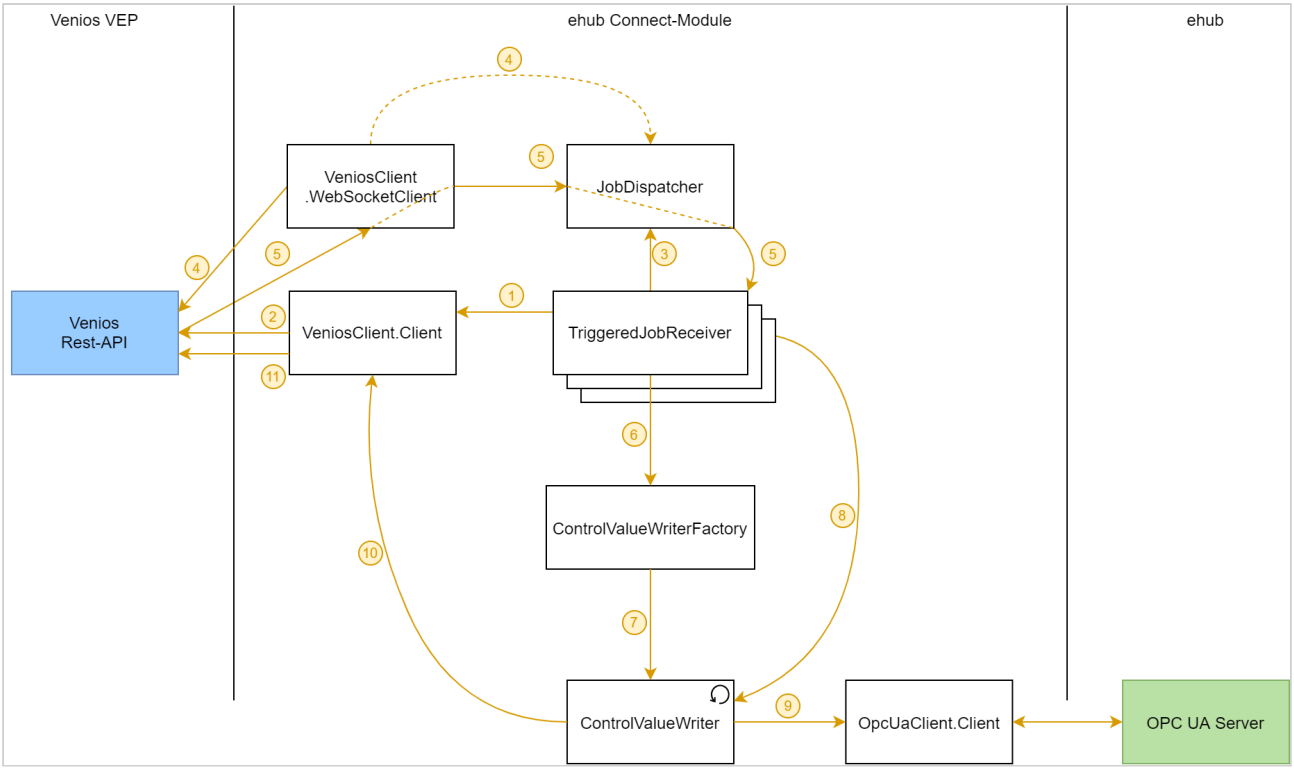
Für jedes Device werden folgende Sampler und Logger instanziiert und jeweils mit einer BlockingCollection<DataSample> verbunden:

Datenpunkte	**Sampler**	**Logger**	**Bemerkung**
device.MeasuredData	PeriodicSampler	TimelineLogger	Sample-Interval = device.MeasuredDataSampleInterval
device.StateData	ObservingSampler	HistoricLogger	
device.ControlData mit Role = None	ObservingSampler	HistoricLogger	

4.2.2.2 Schreiben eines Steuerwertes

Workflow: Setzen Steuerwert

1. TriggeredJobReceivers<WriteControlValueJob<T>, ControlValueWriter>, mit T = {bool, int, float}, abonnieren bzw. WriteBooleanControlValueJob, WriteIntegerControlJob, WriteFloatControlJob mit VeniosClient.Client.Subscribe<...>()
2. VeniosClient.Client sendet Request apiV3/Events/subscribe/WriteControlValueJob
3. TriggeredJobReceivers füllen registry vom JobDispatcher mit TriggerRunnersForJob Actions (werden bei JobDispatcher.Dispatch aufgerufen)
4. VeniosClient.WebSocketClient verbindet den WebSocket und definiert JobDispatcher.Dispatch als Action bei Job Empfänge
5. Job Empfang: VeniosClient.WebSocketClient ruft JobDispatcher.Dispatch mit dem Job auf
6. TriggeredJobReceiver übergibt empfangene Jobs an ControlValueWriterFactory
7. ControlValueWriterFactory erzeugt aus dem Job einen ControlValueWriter
8. TriggeredJobReceiver triggert die parallele Ausführung des Jobs mit ControlValueWriter.Run()
9. ControlValueWriter schreibt den Datenpunkt
10. ControlValueWriter meldet gemäss Resultat der Datenpunkt-Schreib-Operation den Event JobSuccessfulEvent oder JobFailedEvent VeniosClient.Client.FireEvent(...)
11. VeniosClient.Client sendet Request apiV3/Events/add



Workflow: ResearchMode

Damit ein Steuerwert vom ehub angenommen wird, muss regelmässig ein Heartbeat (bzw. Watchdog) geschickt werden und der ResearchMode für das Device auf true gesetzt werden. Das ConnectModul macht genau das, wenn

es regelmässig einen ResearchModeControlJob von Venios erhält. Sobald kein ResearchModeControlJob mehr geschickt wird, wird der ResearchMode auf false gesetzt und es wird kein Heartbeat mehr an den ehub geschickt.

Für Empfang und Ausführung von ResearchModeControlJob existiert ein analoger Ablauf, wobei

- ein JobReceiver<ResearchModeControlJob, ResearchModeTrigger> abonniert und empfängt ResearchModeControlJob
- eine ResearchModeTriggerFactory erzeugt einen ResearchModeTrigger, welcher den für das Gerät verantworlichen ResearchModeKeeper kennt
- der ResearchModeTrigger ruft in der Job-Ausführung im wesentlichen ResearchModeKeeper.KeepInResearchMode() auf

4.2.2.3 Weitere Doku (Klassendiagramm und Flussdiagramme)

4.2.3 ESI ConnectModule: Software Design

4.2.3.1 Data Model

The current data model is shown in the following class diagram:

4.2.3.2 Recording data points

Workflow: Main watchdog

For ESI to release reading values, we must mirror their watchdog signal if we are connected to venios. For more specification details, see https://tools.scs.ch/wiki/x/IgKlBQ.

The main watchdog mirror consists of 2 independent polling parts: one that checks if we are still connected with Venios, and one that reflects the watchdog signal. The polling intervals can be set in `appsettings.json` , respectively
via `MainWatchdog_VeniosConnection_PollingIntervalInMs` and `MainWatchdog_Mirror_PollingIntervalInMs.`

Polling Venios Connection:

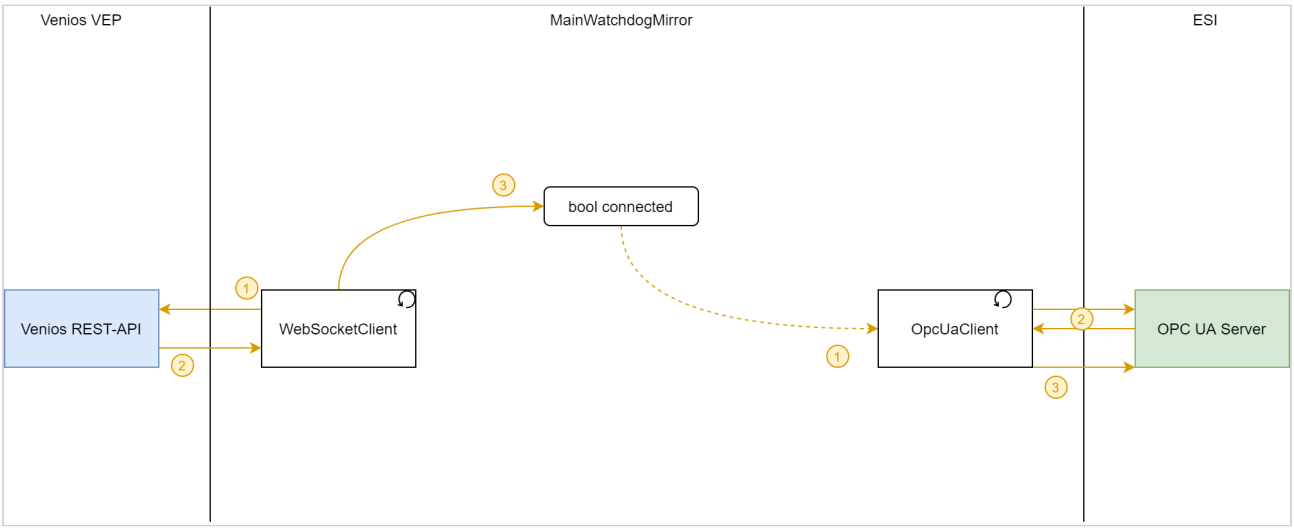
The polling interval is defined by the variable `veniosConnectionPollingIntervalMs.`

1. WebSocketClient (implements IVeniosConnection Interface) pings the Websocket
2. the WebSocket answers and ...
3. the results is written in the `connected` boolean variable

Polling Mirror:

The polling interval is defined by the variable `watchdogPollingIntervalMs.`

1. read `connected` variable (if false: stop at point 1.)
2. read the MainWatchdogEsi signal
3. write back the value to MainWatchdogRemap signal



Workflow: record

The same workflow as the one for the Ehub Connect Module is used, see https://tools.scs.ch/wiki/x/XIvmB.

Concerning measurement signals (CEDA/Real):

The connect module does not differentiate between CEDA or Real datapoints. However, such a signal always has an OPC UA path that ends with ".Value", and has a corresponding boolean signal ending with ".Access". When reading a measurement signal (i.e. signal ending with ".Value"), the OpcUaClient.ClientEsi looks for the corresponding ".Access" signal. If the access signal is false, the datapoint is not added to the return list. If the access signal is true, the signal is added to the return list as expected.

4.2.3.3 Writing control values

Workflow: ResearcherOnline

To write control values, we must additionally mirror the experiment watchdog signal if the researcher is still connected.

The experiment watchdog consists of 2 parts: one that update the timestamp `dateTimeOfLastJob` when a `ResearchModeControlJob` is received, and one that reflects the signal periodically if the timestamp is new enough. The connection timeout and the polling interval can be set in `appsettings.json`, respectively via `ExperimentWatchdogs_ConnectionTimeoutInMs` and `ExperimentWatchdogs_Mirror_PollingIntervalInMs`.

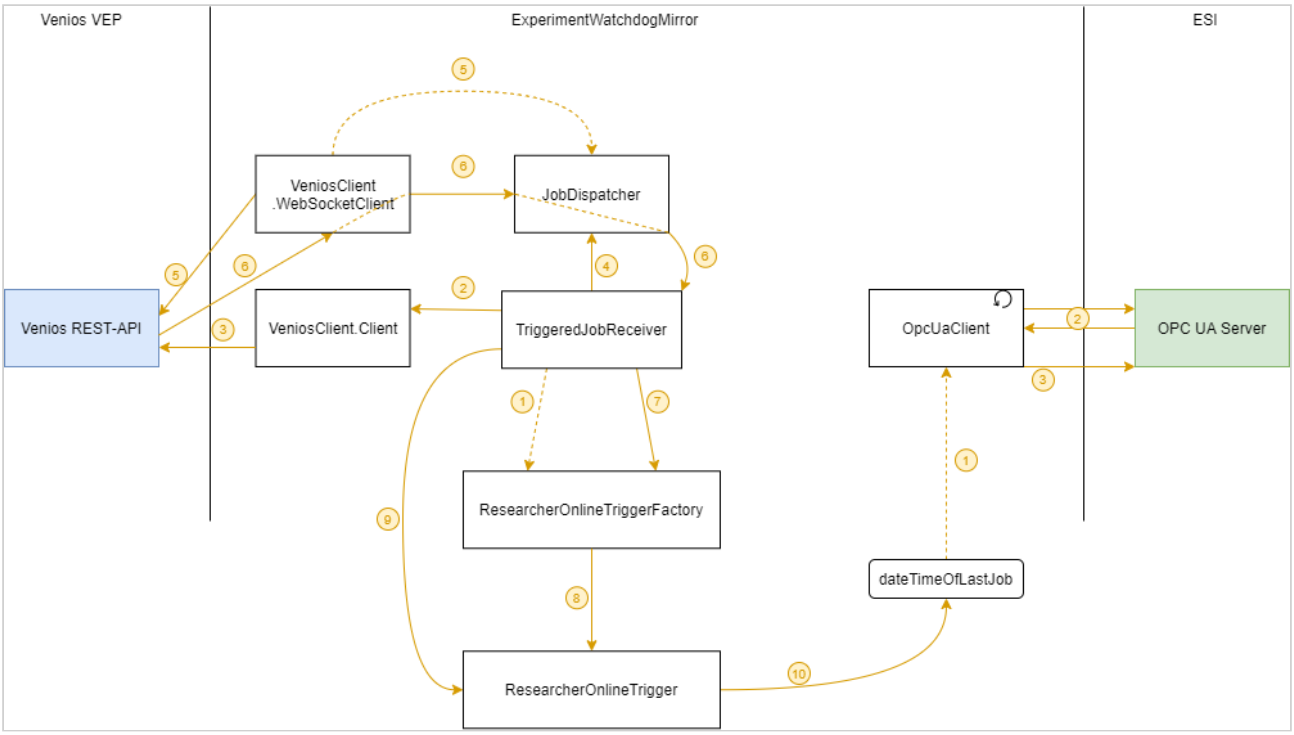
ResearchModeControlJob:

1. TriggeredJobReceiver<ResearchModeControlJob, ResearcherOnlineTrigger> creates a ResearcherOnlineTriggerFactory
2. TriggeredJobReceiver subscribes to ResearchModeControlJob via VeniosClient.Client
3. VeniosClient.Client sends request "apiV3/Events/subscribe/ResearchModeControlJob"
4. TriggeredJobReceiver adds TriggerRunnersForJob Action to the JobDispatcher's registry
5. Connect VeniosClient.WebSocketClient connects the WebSocket and sets the OnMessage Action of the WebSocket to JobDispatcher.Dispatch

6. **Incoming ResearchModeControlJob**: WebSocketClient calls JobDispatcher.Dispatch, which in turn calls TriggerRunnersForJob
7. TriggeredJobReceiver forwards job to ResearcherOnlineTriggerFactory
8. ResearcherOnlineTriggerFactory creates ResearcherOnlineTrigger
9. TriggeredJobReceiver creates and run ResearcherOnlineTrigger as a new Task
10. ResearcherOnlineTrigger updates the DateTime `dateTimeOfLastJob`

Polling Mirror:

1. Check that `dateTimeOfLastJob` is not older than `connectionTimeoutInMs` (if it is older: stop at point 1.)
2. read the ExperimentWatchdogEsi signal
3. write back the value to ExperimentWatchdogResearcher signal



Workflow: write

The same workflow as the one for the Ehub Connect Module is used, see https://tools.scs.ch/wiki/x/XIvmB.

7.3 Class definition ReSIM

See File: CoreClassesDetailedDocumentation.pdf

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

CoreClasses Documentation

1 CommChannel

The **Communication Channel** class handles transfer of signals/non-physical information within a system. It is primarily meant for the communication of measurements and control & feedback signals. Each system and subsystem is automatically initialised with a standard CommChannel instance, but specifying a customized CommChannel object in the system definition of the input file is also possible. The CommChannel collects and contains all the signal information of the system within which it is placed, meaning all local physical, control or exogenous objects but not subsystems. Output signals provided by objects within the scope of a CommChannel are included in the 'com_bus' dictionary under the ['local']['output']['obj_id'] keys, where 'obj_id' is the ID of the object where the signals originate. The output signals are included by establishing a reference connection with the object's .output_signals dictionary using the BasicSystem.connect_objects_to_com_bus method and are hence automatically updated as soon as the computeState (or computeAction or computeValues) method of the respective object has been executed. During execution of the CommChannel, the signal values from the 'local' key (com_bus['local']['output'][obj_id][signal_name]['signal']) are transferred to the 'global' section of the bus using copy.deepcopy. (Listing a custom method under com_bus['local'][signal_name]['method'] is currently not supported.) Signals are thus instantly made available to other local components but are only processed and propagated globally once the CommChannel has made its step at the end of the system's makeStep method.

The com_bus dictionary also has an 'input' field in both the 'local' and 'global' sections. Input information is also grouped by components under the respective 'obj_id' key. Here, signal connectivity information is stored within the 'signal_path' key for each input signal. This information is retrieved from the object's .input_signals dictionary and is used in a later step to retrieve the specified signal from its origin and to connect the appropriate fields by reference. The signal_path is to be specified as a string of component IDs in the format of 'rootID.systemID.objectID.signalName' and can either be absolute (starting at the top-level, i.e. 'root', system) or relative (starting with a local subsystemID or objectID).

Requirements/Assumptions:

- To have access to signals without delay, components need to be placed within the same system and at the same level.
- CURRENTLY NOT SUPPORTED: Methods are supplied for each signal individually, no default is set but it is recommended to pass copy.deepcopy, if no special processing is required. If the signal should not be processed at all (i.e. not copied to the ['processed'] field of the com_bus), the CommChannel.no_op method can be specified.
- Methods must have precisely one required input (the signal value) and one output (the new "processed" signal value).

Open Questions/Needed Functionality:

- During signal propagation, filters or noise etc. could be applied to individual signals.

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

2 CommunicationTree

The CommunicationTree class serves as a container for the CommunicationTreeNodes that handle connectivity and signal communication throughout the model. Its only attribute, called 'root', binds the root level CommunicationTreeNode object. The CommunicationTree class is instantiated by the __init__() method of the Simulation class and initiates the procedure to establish reference connections between input and output signals in the CommunicationTreeNodes when its __init__() method is called.

Requirements/Assumptions:

- The 'root' argument passed to __init__() has to be of type CommunicationTreeNode otherwise a TypeError is raised.

3 CommunicationTreeNode

CommunicationTreeNodes incorporate CommChannels into the communication tree structure and contain links to their respective parents and children within the communication tree. They are mapped one-to-one with CommChannel objects (i.e. one per system/subsystem) and reference the com_bus['global'] key in their '.signals' attribute and the ['local'] key in their '._local_signals' attribute.

3.1 __repr__ (method)

The CommunicationTreeNode's string representation is its ID, which is identical to the system ID of the system it is connected to.

3.2 add_child (method)

If the input 'node' is of class CommunicationTreeNode, it is added as a child of the current CommunicationTreeNode node and and the child's 'parent' property is set to reference the current node. Raises a TypeError if the 'node' is not of class CommunicationTreeNode.

3.3 iterate_post_order (method)

Returns an iterator that traverses the tree in post order.

3.4 goto_root (method)

Returns a reference to the CommunicationTree's root node.

4 BasicSystem

4.1 Create_connector_bus

Create_connector_bus creates appropriate input fields for every component within the system in the system's local connector busses at initialisation. Fields are created in *self.Connector[connector_type].bus['connected_dn'].* The component's output fields (assumed to be energyCarrier objects) are connected by reference. This allows both the connector and the physical component (from where the physical quantity originates) to change and extract information without the need to copy or update.

Requirements/Assumptions:

- Connected fields must be energyCarrier objects matching the connector's type
- Components must contain a dict of form *{'connector_type': [energyCarrierObj(s)]*

    - The energyCarrier objects are initialised with user-supplied initial values during instantiation of the components
- Components must have a unique .name attribute that will be used as a key in the connector bus

Open Questions/Needed Functionality:

- Do the connector bus's input fields need to be propagated to a higher level or is it enough to propagate the output?
- Other components within this subsystem can be connected to the Connector's input field (by reference) to receive the physical property without delay (I.e. for sequential execution and direct dependence)

4.2 register_id (method)

The register_id method enforces that any new id to be registered conforms to the following three rules:

1. All IDs must be of type string.
2. Empty strings are not valid IDs.
3. IDs must be unique within a system (not including its subsystems)

If one of the criteria is not met, an error is raised. (A TypeError for the first criterion and a ValueError for the other two.) Each system object maintains a set of all its registered IDs against which any new IDs are checked for uniqueness. If they pass the check, they are then added to the set.

5 MatlabInterfaceMixin

The MatlabInterfaceMixin class is to be used as a parent/mixin class for all component and control classes interfacing with MATLAB/Simulink. These child classes also need to inherit from the appropriate core class, either Component or Control. The MatlabInterfaceMixin class mainly adds some functionality to the __init__ method: It imports the matlab.engine module and instantiates a MatlabEngine, if none exists already, which it then binds to the 'config' module to make it available globally. In addition to this, it also allows a (relative or absolute) path of the location at which the required MATLAB scripts and models are stored to be specified. This path is then added to the MATLAB path. In addition, the mixin provides the following methods for use in child classes:

5.1 is_matlab_fcn (method)

Check whether file name represents valid MATLAB functions. A warning is raised if a script is passed instead. For any other form of .m file, or if the file can't be found, an exception is raised.

The method takes a 'matlab_file_name' (string) as input and returns a Boolean.

5.2 run_matlab_init_fcn (method)

This method is used to run the user-defined (MATLAB) initialisation function that has been specified using the 'init_fcn_name' simulation parameter. Before executing the .m initialisation function, the method calls 'is_matlab_fcn' to make sure the supplied 'init_fcn_name' refers to a valid MATLAB function. It then binds the function to 'self.matlab_init_fcn' so that it can be called directly, without using the self.eng() method.

The method has no inputs or outputs.

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

6 NamedClass

This class is intended to provide a unique .name property to its subclasses. Exogenous, Physical and Control inherit from it. Systems may also be given .name properties in the future.

6.1 name (property)

The **.name** property is set during instantiation of subclass objects, either with a user-supplied string or automatically based on the class name if no string is supplied. In this process, it is checked against the set of existing names and if it is not unique, a ValueError is raised. If the user-input is not of type string, a TypeError is raised. After it has been set, the .name property cannot be changed.

Much of the functionality of the .name property has since been taken over by the .id attribute. The .name property's main remaining use case is for defining physical connections in connectors.

Requirements/Assumptions:

- The user input must be of type string
- The name must be unique

6.2 Automatic_name (method)

In case the user has not supplied a name, the **automatic_name** method is called to assign one automatically. The current implementation works as follows:

1. A name is generated using f'{type(self).__name__}_{counter:02d}', where the counter is initially 1
2. The generated name is checked against the set of existing names. If an error is thrown (i.e. the name already exists), the counter is increased by 1 and the function is called recursively.

Open Questions/Needed Functionality:

The current implementation is perhaps not the most elegant. If a large number of instances from the same class need an automatically assigned name, then the function would have to do a lot of brute-forcing to assign all the names. For n instances, the function would be called n^2 times and would fail to assign a name n*(n-1) times. The performance impact of this might have to be analysed and alternative options investigated.

Alternative implementation options might be:

1. Finding existing fields including the same class name in the ._names set using regular expression.
    a. Count the number n of found fields and set the counter to either n or n+1
    b. **Or** find the largest assigned counter (also using regex) and use max counter+1 for the new name
2. Maintain a dictionary that keeps track of all the automatically assigned names for each class (i.e. {className: counter}, check whether the class name of the currently processed instance already exists in the dict. If it does, increase the counter by 1 and use it to generate the new name. If it does not, create the field with counter set to one and assign the name using this counter.
    i. Dict = {'className': counter}
    ii. If type(self).__name__ in dict:
    iii. Dict['className'] += 1
    iv. Else:

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

v. Dict[‘className’] = 1
vi.
vii. Name = f’className_{dict[‘className’]}’

7 TimeKeeping

The TimeKeeping class is intended to keep track of time throughout the simulation in a coordinated and consistent way. It’s properties and methods are initialised according to the user’s need for real or non-real time automatically. Thus, no explicit distinction between the two cases is necessary during simulation.

7.1 dt (property)

The time increment, dt, is a property of the TimeKeeping class based on the private attribute _dt. There is no getter method, when dt is called, _dt is returned. The setter method is dependent on the real/non-real time setting of the simulation. For non-real-time simulations, _dt is initialised to match the chosen simulation timestep and the setter method does nothing. For real-time, the setter method updates _dt using the difference between the input parameter ‘current_time’ and the TimeKeeping class’ previous_time property. The setter method is intended to be used only once per timestep by the Simulation class.

Requirements/Assumptions:

- Since ._previous_time is updated at the beginning of each step, .dt is based on the duration of the preceding step.
- dt should only be set in the Simulation class
- Other objects should not inherit from TimeKeeping but access dt using config.dt where config is the config module used to share global variables across the entire code.

7.2 previous_time (property)

The previous_time property is intended to keep track of the time before the current timestep was executed and is used for calculating the dt property. It is updated at the beginning of every timestep. At the core of the previous_time property lies an attribute called _previous_time, which is directly returned when previous_time is called (no getter method is implemented). Like dt, previous_time’s setter method takes the current time as an input argument. The current time input is then stored as _previous_time. The property should also only be set once per timestep.

8 Simulation

The Simulation class inherits from TimeKeeping and is meant to provide the simulation environment that wraps around the model. It provides methods for running a simulation (in real / non-real time), saving results, printing information to the console and more. It is also to be used as a source of timing information (according to the methods and properties given in TimeKeeping). While the class provides methods for saving results (save_results) and updating the console (_update_console), their implementation is case specific and as such, they are meant to be overridden in a case specific subclass.

Open Questions/Needed Functionality:

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

- In future, methods like save_results and _update_console should have a more generalized implementation with an increased degree of automation such that they need not be overridden for case specific applications. A user should simply be able to provide a list of variables/properties to be saved/printed to the console as part of the input parameters.

8.1 run (method)

The run method is initialised using either the definition of _run_real_time or _run_non_real based on the user supplied Boolean 'is_real_time'. In either case, an optional start time argument (in the datetime format) can be supplied, and the model will be simulated using the specified timestep and simulation duration. The _update_console method is used to print information to the console after every iteration. Using the keybord interrupt (ctrl + c), the simulation can be prematurely terminated while still ensuring external connections are properly disconnected and results are being saved (using the save_results method). Stopping the code using the IDEs stop button may, however, cause the simulation to be stopped improperly in which case results will be lost.

8.2 log_to_csv (method)

log_to_csv provides a method for writing logged data to a csv file.

Requirements/Assumptions:

- Log_list should be a nested list with at most two levels (i.e. a matrix with two dimensions) and must not contain any dictionaries
- The log_header argument should contain one header/description for each variable in the log_list argument
- The 'clear_log' option must not be used with power flow results (from Adaptricity) as it would alter the structure of the pf_results_log. If it is desired to clear the pf_results_log, the pop_all method should be used to assign the power flow results to a new list, which can then be passed to the log_to_csv method with clear_log=False

Open Questions/Needed Functionality:

- Distinguish between log_list inputs that need to be rearranged from grouped by variable to being grouped by timestamp (For power flow results, the _save_pf_result_worker method already regroups the logs while this is not the case for scalar result logs that are grouped together to be saved to a single csv).

8.3 save_pf_results_worker (method)

The save_pf_results_worker method is meant as an intermediate between the save_results and log_to_csv methods whenever power flow results (from Adaptricity) need to be logged. It unpacks the power flow results from a list of dictionaries and converts it into nested lists (one list per variable) containing results for each timestep and element (i.e. node, branch, transformer, etc.). Each list is then passed to the log_to_csv method and written to a file. Method written by Raphael Wu (RRE, ETH Zürich)

Requirements/Assumptions:

- The log_list argument should be a list of dictionaries where dictionaries are in the node_dict/branch_dict, etc. format and every dictionary in the list contains the results for a different timestep (in the order in which they appear in the list)

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

9 SimulatedObjects

The SimulatedObjects class is to be used for definitions that need to be shared across Control, Exogenous and Physical classes, and as such is a parent to the Physical, Control, Exogenous and Connector classes as well as their children. SimulatedObjects itself inherits from NamedClass.

9.1 id (attribute)

The SimulatedObjects class sets the .id attribute for all its children.

Requirements/Assumptions:

- The ID must be of type string
- Empty strings are not valid
- The ID must be unique within the system (but not necessarily it's subsystems)

9.2 update_inputs (method)

The update_inputs method, implemented in the SimulatedObjects class, is key to the input signal handling of all child classes. It must be called at the beginning of every computeState, computeAction and computeValue method to properly update and process the input signals for the respective class.

By calling the update_inputs method, input signals are retrieved from com_bus dictionary specific to the local system by first copying com_bus['global']['input'][self.id], where self.id refers to the ID of the object who's method is being called, and then updating the resulting dictionary with the contents of com_bus['local']['input'][self.id], meaning that signals from 'global' that also exist in the 'local' portion of the com_bus will be overwritten. This is done because the 'local' signals are updated immediately whereas the 'global' signals are only updated at the end of the system's makeStep method, after all the physical and control classes have made their steps. The resulting dictionary is bound to the self.inputs attribute. While child classes may overwrite the subsequent steps of the update_inputs method, this first part should generally be maintained.

From the self.inputs dictionary, the signals are then set as attributes of the class using the setattr function where the dict key is set as the attribute name and the value stored in self.inputs[key]['signal'][0] is bound to said attribute. The dictionary keys are identical to the user supplied keys in the input_signals parameter of the input json file.

Requirements/Assumptions:

- Any variable name (key in the communication bus: com_bus['global'/'local']['input']['obj_id'][key] must match an existing attribute (variable) of the class, otherwise an AttributeError is raised
- If an attribute/variable is designed to be updated using the update_inputs method, but no input signal was defined in the json input file, no error or warning will be raised, and the value will remain constant

9.3 _preprocess_cfw_signals (method)

Thanks to the _preprocess_cfw_signals method, input signals that are exclusively intended to be used when the useCFW parameter is set to true (HIL simulations or reading live data from the Venios platform), can be defined using the cfw_signals key in the input_signals dictionary. It is not meant to be called explicitly from child classes since it will be executed during the __init__ method of SimulatedObjects and is thus covered by calling super().__init__(…) on any child classes.

Created on: 16.03.2021

Last modified: 31.10.2022, by Philippe Buchecker

If the cfw_signals key exists within the input_signals dictionary, it will be popped and if useCFW is set to true, it will be passed to the _preprocess_cfw_signals method which processes the signals and adds them back into the input_signals dictionary one by one. If cfw_signals does not exist, a warning is raised and if it exists but useCFW is set to false, it will simply be discarded. So far, this all happens within the __init__ method, within the _preprocess_cfw_signals method the type of cfw_signals is checked (non-empty dictionary) before converting all '.' characters in the signal path to '_' characters, adding the signal source and adding the signal back into the input_signals dictionary.

Requirements/Assumptions:

- cfw_signals must be a dictionary with the same format as the regular entries in the input_signals dictionary (key = variable name, value = signal source), but in this case, the signal source is to be specified without the object ID since this will automatically be added at runtime based on the general simulation settings.

7.4 User guide ReSIM

See File: Simulation Framework User Guide.pdf

Created on 30. 05. 2022 by Philippe Buchecker (bucheckp@fen.ethz.ch)
Last modified on 29. 11. 2022 by Philippe Buchecker (bucheckp@fen.ethz.ch)

Simulation Framework User Guide

Contents

Required Files

The ReMaP Simulation Framework (SFW) is an open-source Python software package that consists of the following files:

- SFW_main.py: This is the main file that needs to be run from your Python interpreter or the command line. It is also here, that the user can specify which input file should be loaded.
- input_parameters.json An example file specifying the input format, can be modified by the user.
- Venios_signals.py: Example file
- ehub_control_example_signals.csv: Example file
- coreClasses.py: A Python file that contains class definitions for the core structure of the SFW.
- modelLibrary.py: A library of standard models that can be used for SFW simulations.
- config.py: File containing variables and configurations shared across the entire codebase.
- config.ini: Contains the base URL needed to connect to Venios
- energyCarriers.py: Defining classes that facilitate the handling of different energy carriers and physical quantities.
- requirements.txt: Listing the packages required to run the SFW.
- common (folder): Contains Control Framework code for data exchange and hardware-in-the-loop via Venios or Empa's OPC-UA interface.
- access.ini: File holding the user's access credentials for any interface or external software they wish to use (Venios, Ehub, Empa Rest API, Adaptricity). (The file is provided as a blank template.)
- tkn.ini: File for specifying the Adaptricity access token. (The file is provided as a blank template.)
- Simulink tank model example: Folder containing a small Simulink model example along with all the required auxiliary files and functions.

Installation Guide

(This guide describes the setup for use with integrated Simulink models. If the SFW is used without Simulink models, steps 4a and 4b can be skipped. If MATLAB is not used altogether, steps 3a, 4, 5 and 8 are also not necessary.)

1. Get the latest SFW code from Adamantios Marinakis (amarinakis@ethz.ch) or Philippe Buchecker (bucheckp@fen.ethz.ch)
2. Move and unzip the code to a location of your choosing.
3. Download python 3 here: https://www.python.org/downloads/windows/
    a. [Optional] You need a 64-bit version to be compatible with MATLAB. Make sure your MATLAB version supports the python 3 release you want to install.
    b. When installing python, select "add python to path".
4. [Optional] Use/install a MATLAB version, which is compatible with the python 3 release you want to use. (Find compatibility information here.)
    a. To run Simulink models in real-time, your installation needs to include the following toolboxes:
        i. MATLAB Coder
        ii. Simulink
        iii. Simulink Coder
        iv. Simulink Desktop Real-Time

   b. Type the following command into your MATLAB command line:
      i. *sldrtkernel –install*
      ii. This installs the MATLAB real-time kernel. You can also find official documentation here
5. [Optional] Install the MATLAB Engine API: (For more information, consult the mathworks help page.)
   a. Type: *matlabroot* (into your MATLAB command window)
   b. Copy the output (MATLAB root directory)
   c. Open a windows command prompt with administrator rights
      i. Press: [Windows Key] + [R]
      ii. Type: *cmd*
      iii. Press: [ctrl] + [shift] + [enter]
   d. Install API:
      i. Type: *cd "matlabroot\extern\engines\python"*
      ii. Type: *python setup.py install*
      iii. (If there is no output following the second command, you may need to use the full python path instead of "python": "*C:/users/**YourUsername**/AppData/Local/Programs/Python/Python38/python.exe*". Depending on your python release, this path may vary.
6. Set up a virtual environment and create a PyCharm project (or use your IDE of choice):
   a. Set up a virtual environment (using your 64-bit, MATLAB-compatible python release) in the folder containing the SFW code:
      i. Navigate to the folder in the file explorer
      ii. Type *cmd* into the address line. This should open a command window
      iii. Type "python3 -m venv .venv" into the command window. You may have to use the full python path instead of python3, especially if you have installed several python versions. The path will most likely be: "C:\Users***YourUsername***\AppData\Local\Programs\Python\Python38\python.exe"
      iv. If the creation of the virtual environment was successful, run the following commands one after another:
      v. ".venv\Scripts\activate.bat"
      vi. "pip install -r requirements.txt"
      vii. For more information, see also the original documentation on setting up a virtual environment: https://docs.python.org/3/tutorial/venv.html
   b. Open the folder as a PyCharm project
7. Test run the SFW code by executing the SFW_main.py script.
   a. If there are no errors, the installation was successful.
8. If you are trying to run the SFW with MATLAB and you encounter errors, try the following:
   a. Copy the folder: "C:\Users***YourUsername***\AppData\Local\Programs\Python\Python38\Lib\site-packages\matlab" into your folder containing the SFW code.
      i. If there is no subfolder named "matlab" in your python path, the MATLAB Engine API did not install properly.
      ii. Try running the SFW again.
   b. Copy the folder: "C:\Program Files\MATLAB\R2020b\extern\engines\python" into your folder containing the SFW code.
      i. Try running the SFW again.

Input Format

The ReMaP SFW does not have a graphical user interface, instead input information is provided via a json file. Users can utilise the input_parameters.json template, that is provided with the SFW. The file has two main sections: “general” and “entire_system”. In the former, general information, such as the simulation duration and the simulation mode, is specified. In the latter, the user can define the system they wish to simulate.

General Inputs

Table 1: General Simulation Parameters

“sim_duration_seconds”	Float.
“timestep_seconds”	Float. Timestep in seconds. If the simulation is not in real-time, there are no limits on the step size. For real-time simulation, the timestep is limited by the calculation time for the model. This may vary based on the model’s size and level of detail. The default is two seconds.
“is_real_time”	Boolean. If true, the simulation will run in real-time, timesteps will be synchronised with the local machine’s clock. If it is set to false, the simulation’s speed is only limited by the computational cost of the model.
“use_control_framework”	Boolean. If true, the SFW will initialise and use the Control Framework (CFW) to communicate with (read and/or write) external hardware according to the settings specified in “ControlFramework”. Any components that can switch between HIL and pure simulation will be switched to HIL mode. The default is “false”.
“devices_not_controlled”	Boolean. If true, the CFW will be read-only and will not request control-access from the hardware. Models can also be set up to ignore control commands in simulation-only mode when “devices_not_controlled” = true and “use_control_framework” = false. The default is “false”. **For HIL mode, “devices_not_controlled” should not be set to true without coordinating with the hardware owners/operators first.**
“save_file_name”	String. Base name for the result files. The full name will start with the time and date at which the files were saved and will also include information about the variables contained within each file.
“results_folder”	String. The user can optionally specify a subdirectory or full path to which the results should be saved.
“ControlFramework”	Dictionary with three required keys: “timeout”, “signal_file_name”, and “input_signals”. See

	below for more information on the format and usage of each key.

Unlike other exogenous data classes, the interaction with the Control Framework is not part of the modelled system and its parameters are therefore given inside the “general” section rather than the “entire_system” section. The “timeout” parameter is used to specify a timeout in seconds. It is only used when the CFW is connecting to Venios and not using the direct interface with Empa[1]. The timeout defines how long the CFW waits for a response from Venios before aborting and moving on with the simulation using the last known value. In this case, it will try reading new values in the subsequent timestep.

The “input_signals” parameter is used to set up communication with the rest of the model according to the rules defined in Defining Communication. It can be integrated into the main input_parameters.json or given as the filename of a separate csv or json file. The signal source should typically be a control signal from a controller class. Unlike elsewhere, the keys in the “input_signals” dictionary do not reflect the variable name inside the class but rather are used to specify either the VeniosID or the OPC Path, depending on the desired interface. Any signals specified in the “input_signals” will be communicated to the hardware as control signals.

The “signal_file_name” parameter is used to select the file that holds the definition of any signals that need to be accessed via the CFW. If the connection is established via Venios, a .py file formatted according to the template given by Venios_signals.py is expected. If a direct connection to Empa is used, the format of ehub_control_example_signals.csv is required.

Defining the System

The topology and execution order of the system is defined through the structure and order of the content assigned to the “entire_system” key. (Subsystems, components, exogenous data, and connectors are each executed in the order they are specified, if there are multiple within the same system.)

For every object, a “className” and “id” key must be specified. The former must be a string containing the name of the class that is to be used to instantiate the object. If the class is not defined in SFW_main.py, the string should be formatted as “moduleName.className” where moduleName is the name of the Python module /file in which the class is defined. The “id” key should be filled with a user-selected human-readable string used to identify the object. It is also used to specify where feedback or control signals should be read from. The id must be unique within the system where the object is to be used but can occur multiple times across several systems.

All components, controllers and connectors expect three input arguments of type dict, which are (in order):

1. Model parameters (model_param)
2. Initialisation parameters (init_param)
3. Simulation parameters (sim_param)

These are collectively specified in the input file using the “param” key. (i.e. “param”: {model_param: {}, init_param: {}, sim_param: {}}) As part of the initialisation parameter dictionary (the second input argument), a unique name can be defined for each component. It should be formatted as a string

[1] For this case, a similar error handling is already implemented at a deeper level.

and included with the key 'name'. If the supplied name is not of type string, a TypeError is raised. The user-supplied names should be human readable and related to the component to which they are being assigned. This will help make the simulation output more readable and will facilitate debugging.

Defining Communication

Most classes, especially controllers or physical components, expect to be given a number of input signals (i.e. feedback or control signals) that are specified in the class's respective doc string. These input signals are used to receive information distinct from physical inputs. They are specified by creating an "input_signals" dictionary in the object's simulation parameters (sim_param). In this dictionary, signals are defined by using the variable name (as given in the class doc string or as used in the computeState/computeAction method) as the key and giving the signal source as the value. The signal source must be given as a string containing at least the source ID and the name of the output signal separated by a full stop (i.e. "sourceID.signalName"). If the signal source is not part of the same system or subsystem, the absolute signal path must be specified starting at the top-level system, which is always named "root_sys". The signal path will then look like this: "root_sys.subsysID.subsysID.sourceID.signalName".

If a class supports switching between pure simulation mode and hardware-in-the-loop (HIL), the signals that are to be retrieved via the hardware interface (using the Control Framework) can be given in a nested dictionary called "cfw_signals" inside the "input_signals". For these signals, it is enough to specify the signal name (the VeniosID or the OPC path if Venios or the Ehub Interface are being used respectively). The signal source is then selected during initialisation based on the simulation mode.

Defining Physical Connections

When using components that have physical inputs or outputs, an appropriate connector class must be defined in the same system. There must be one connector object for each type of energy carrier that is being used. Physical outputs are then automatically connected to the connector with the same "type". Physical inputs typically require that the user specifies the key name that should be used to read from the connector's output. Output keys connectors create during initialisation usually consist of the connector's name and a two-digit counter separated by an underscore.

Unlike signals, which can be exchanged by two objects anywhere in the entire system, physical connections adhere to the hierarchy of the system and can only connected to other objects in the same subsystem. The connector objects then handle physical connections to higher and lower levels in the system hierarchy.

Execution Order

Execution of the model starts through the top-level (root) system's makeStep method but takes place bottom-up (depth-first post-order). Before any of the system's local elements (exogenous data, controller, components, etc.) are called, the makeStep method of all subsystems (in the order they are defined in the input file) is called.

Within a system, the execution order is as follows: Subsystems, Exogenous (in order of definition), Components (physical, in order of definition), Control, Connector (in order of definition).

Loading Exogenous Data

Exogenous data (e.g. PV production or demand timeseries) can be loaded using one of the Exogenous classes. It is recommended to provide exogenous data in a CSV format where the first column is named "timestamp" and contains the date and time of the data samples. CSV files that

follow this format can very easily be loaded using the ExogenousCSV class and the following simulation parameters:

Table 2: Exogenous Data Parameters

"data_path"	String. Subdirectory or full path to data file location.
"data_filename"	String. Name of the data file including file extension.
"data_columns"	List[String]. A list of column headers that should be loaded. The column names that are not listed will not be loaded into the SFW. If the list is empty, all data columns are loaded.
"lookup_file"	String. Name of a csv file containing a lookup table to change data headers. Data will be available throughout the simulation using the name from the column headers.
"start_index"	Integer. Specify the row at which data should start being read. Default is 0.
"interpol_limit"	Integer. Specify the maximum number of data samples that are interpolated if a column starts with missing values. Missing values that occur after the first viable sample are padded.

Using External Software

The Simulation Framework supports the integration of models and algorithms from external software. Namely, plug-and-play functionality is offered for the Adaptricity power flow solver as well as MATLAB and/or Simulink.

Adaptricity

Adaptricity is a commercial power flow software that requires a licence to use. If you do not already have a licence, please contact either Adamantios Marinakis (amarinakis@ethz.ch) or Philippe Buchecker (bucheckp@fen.ethz.ch).

Before you can use Adaptricity through the SFW, you must give your access credentials in the appropriate places:

- Add your username and password (outer login) to the access.ini template in the "access_adaptricity" section.
- Add your token to the tkn_adap.ini template. (To create the token, log on to the Adaptricity web platform and navigate to User → MyProfile and then click on the *Create Token* button.)

The SFW can be set up to run power flow calculations via Adaptricity's Rest API by including the modelLibrary.NetworkAdaptricity connector class (or one of its children) in the system. Loads and generators can then be added to the grid by specifying the Adaptricity bus ID when initialising the electricity energyCarrier object that defines the output of electrical components in the system. Beware that Loads and generators which are defined in the Adaptricity grid model will be included in every power flow calculation, if the "Connected" checkbox is ticked.

The following simulation parameters (sim_param) will always be required for the Adaptricity interface, more may be needed depending on the chosen connector implementation and functionality:

Table 3: Adaptricity-Specific Parameters

"project_name"	String. The name of the Adaptricity.Sim project to be used.
"grid_model_name"	String. The name of the Adaptricity.Sim grid model on which to run power flows.
"maxIter"	Integer. The maximum number of power flow iterations on Adaptricity.Sim, for more information consult the Adaptricity web platform. The default is 1000.
"tolerance"	Float. Sets the convergence error for power flow calculations, for more information consult the Adaptricity web platform. The default is 10^{-6}
"slack_voltage_pu"	Float. The voltage at the slack bus in per units. The default is 1.
"access_file_adaptricity"	String. Name of the .ini file that contains the Adaptricity access information, usually "access.ini".
"token_file_adaptricity"	String. Name of the .ini file that contains the Adaptricity token. The default is "tkn_adap.ini".

MATLAB/Simulink

To utilize MATLAB/Simulink code within the SFW, a few extra steps need to be followed during the setup of the SFW codebase. More information on these steps is given in the Installation Guide section. For the usage of Simulink, the following MATLAB toolboxes are required: MATLAB Coder, Simulink, Simulink Coder, and Simulink Desktop Real-Time, these are, however, not necessary to run regular MATLAB functions via the SFW.

Any MATLAB code you wish to integrate into the SFW, whether it's a control algorithm, a model, or a Simulink model, should be interfaced via a MATLAB function. This is especially true for control algorithms and models, where signals or physical outputs need to be returned to Python. Optional initialisation functions, that are to be called during the initialisation of the SFW can technically also be MATLAB scripts[2], if they do not require any feedback to Python. Initialisation functions can be specified using the "init_fcn_name" simulation parameter.

The SFW modelLibrary provides three base classes to work with MATLAB: MatlabComponent, SimulinkComponent, and MatlabController. The specifics of each class are described in the following.

MatlabComponent

This class is designed to accommodate models of physical components that are written in MATLAB. The name of the MATLAB function that contains the model can be set using the "model_fcn_name" simulation parameter. It is then called using the run_matlab_model() method, which is designed to be overwritten for more flexible input/output handling. See Integrating Python Models and Writing Customised Classes for more details on how to do that.

[2] This will, however, prompt a warning message.

MatlabController

The MatlabController class offers similar functionality as the MatlabComponent class but is geared towards control algorithms. The MATLAB control algorithm is given using the "ctrl_fcn_name" simulation parameter and to run it, the run_matlab_controller() method is called. For control algorithms that are computationally more expensive (e.g. because they solve an optimisation problem) the MatlabOptimalController class can be used instead. It is set up to run the control algorithm asynchronously using the background=True parameter when calling the MATLAB function. During each iteration of the SFW, the controller class checks whether the MATLAB optimisation has finished and new setpoints are available. If that's the case, it will update the setpoints and restart the optimisation. If not, the old setpoints will be reused.

SimulinkComponent

To integrate a Simulink model, a function handling the input/output must be written in MATLAB and specified using the "io_fcn_name" simulation parameter. In addition, the Simulink model name must also be declared using the "Simulink_model_name" simulation parameter. The name will be used to directly interact with the Simulink model and is also defined as the unique "name" property of the SimulinkComponent instance. This is done deliberately to prevent two separate class instances from trying to run the same Simulink model in the same simulation.

The SimulinkComponent class only supports simulations in real-time. This requires a Real-Time Synchronization block to be included in the Simulink model. For more information on setting up your Simulink model for real-time simulations, consider the Simulink Desktop Real-Time documentation or look at the Simulink tank model example that is provided with the SFW. The "read_write_tank.m" io-function example shows how to use constant-blocks as inputs and a DataView block to collect outputs.

Integrating Python Models and Writing Customised Classes

It is part of the SFW's design philosophy that existing models can be adapted to better suit the user's needs and that researchers can incorporate their own (python-based) models and algorithms into the software. Integrating Python models and writing custom classes does, however, require some base knowledge of Python and familiarity with object-oriented programming.

Any models from the SFW's model library can be adjusted by defining a new class that directly inherits from the original model. Researcher models and algorithms can also be included by defining a new class that inherits directly from either the Component, Control or Connector class, depending on their type. Furthermore, the Exogenous or ExogenousCSV classes and the Simulation class can be modified. The former to accommodate the import of exogenous data that cannot easily be converted into a format that is compatible with the existing data import structure and the latter, to modify how results are being saved and what information is printed to the console at runtime. Apart from _update_console() and save_results(), no other methods of the Simulation class should be altered. Any other classes, particularly in the coreClasses module, should not be changed at all.

Defining New Classes

New classes should be defined either within SFW_main.py or in a new project-specific python file inside the project folder. If the latter option is chosen, the file must be imported into SFW_main and class names in the input_parameters.json file must be preceded with the name of the project-specific python file (i.e. "fileName.className").

Any new class, whether Component, Controller, Connector or Exogenous, must have two methods that adhere to a specific signature and structure: The __init__ method, that is used to initialise the

model at the beginning of the simulation and a method to execute the model or control algorithm defined within the class. The latter is called computeState, computeAction, runConnector or computeValue, depending on the type of class. The methods can either be completely overwritten by redefining them in the new class or given additional functionality by including a call to the original method using the "super().method_name()" statement in the new definition.

The __init__ Method

This method is called during the initialisation of the model at the start of the simulation. It is used to read in the input parameters, define initial values and set up both physical outputs and signal communication. If the __init__ method is to be modified in a customised class, it must include a call to the __init__ method of its parent class using the statement:

```
super().__init__(model_param, init_param, sim_param)
```

This is necessary because a lot of the functionality that is used to automatically set up the simulation comes from parent classes and works through the inheritance of the __init__ method. Whether the call to super() is placed at the beginning, at the end or even in the middle of the new __init__ method definition is entirely dependant on what functionality a user wants to change.

Defining Parameters

Additional model parameters can be defined in the __init__ method by reading keys from one of the parameter input dictionaries (model_param, init_param, sim_param) and assigning them to a class attribute, e.g.

```
self.parameter_name = model_param.get('parameter_name', default)
```

There is no fundamental difference in the functionality of these three input dictionaries, they simply allow users to group parameters according to their nature (i.e. physical parameters of the model, initial values, parameters for simulation mode and connectivity). Accessing the keys with the get() method allows users to specify a default value that will be used if the key is not defined in the dictionary. If no default value is defined, it will be set to None. Be aware that there will be no errors or warnings if a parameter is missing in the input file. If you wish to override the parameter definition from the parent class, your new definition must come after the call to super().

Defining Outputs

Output signals and physical outputs must be defined within the __init__ method for them to be properly connected to other parts of the model. Both the "output" (physical output) and "output_signals" attributes are of type dictionary. If you whish to overwrite their definition from the parent class, you can assign a new dictionary to the attributes after the call to super(). If you want to add new entries to the existing dictionary, only define the individual keys. Physical outputs must follow the convention key = output type (e.g. 'Electricity' or 'Hydrogen'), value = list(energyCarrier object). Output signals follow a different convention and should be defined using key = variable name, value = list(value), where the list usually contains only one element which can be another list or dictionary. During the simulation, new values must be set by changing either the energyCarrier properties or the list element. Setting a new list or energyCarrier object will break the communication / connection.

Defining Inputs

To add additional input signals, it is generally sufficient to define them in the input file by adding them to the object's "input_signals" parameter according to the rules defined in the section Defining

Communication and initialising the parameter in the __init__ method using the same variable name that is specified in “input_signals”.

computeState, computeAction, computeValue or runConnector

This is the method that gets called during each step of the simulation and runs the model/control algorithm, its name changes based on the type of class and Table 4 shows how the two correlate. The method contains three main activities that are, in order: updating the input signals and physical inputs, executing the model or control code, and updating the object’s outputs (both physical outputs and signals).

Table 4: Correlation between class type and method names.

Class Type	Method Name (input arguments in brackets)
Component	computeState(self, com_bus, connector)
Control	computeAction(self, com_bus)
Exogenous	computeValue(self, elapsed_time, step)
Connector	runConnector(self, com_bus)

Updating Inputs

The input signals are updated using the update_inputs method that is inherited from the core classes. The update_inputs method must be given the communication bus dictionary (com_bus) as an input argument. It then extracts the relevant input signals from the com_bus and stores them in the “inputs” attribute. “inputs” is a dictionary with the following structure:

```
self.inputs = {'variable_name': {'signal': [value]}}
```

After creating the “inputs” dictionary, the method creates class attributes from the dictionary contents with self.variable_name = value. This allows the input signals to be seamlessly used within the model. If necessary, the second step of the update_inputs method can also be adjusted to process and group input signals according to the user’s needs.

Physical inputs need to be retrieved explicitly using:

```
self.variable_name = connector['connector_type'].bus['connected_up']
                     [key][0].variable
```

where “variable” refers to the energyCarrier object’s property that is of interest (often, this will be either “power” or “mass”) and the key is defined according to the section Defining Physical Connections.

Exogenous classes do not support inputs, from other parts of the model. Their computeValue method generates outputs solely based on the information and exogenous data it received from the input file during initialisation (this can, however, include instructions to call an API and receive new data from there, for example).

Executing the Model/Algorithm

The actual model or control algorithm can be defined inside the computeState/computeAction method, but it is recommended (especially for models) to place the code inside a separate method (e.g. run_battery_model()) that is then called by the main computeState method. This makes it easier to apply changes when customising a model, since it may then be enough to change the model method and leave the computeState method as is (given that the outputs do not change).

Updating Outputs

At the end of the computeState/computeAction method, the physical outputs and output signals must be updated. It is not possible to simply bind a new dictionary to “ouputs” or “output_signals” since that would break the connection. Thus, every key of the two dictionaries has to be updated separately, making sure to change the energyCarrier property or the list element rather than binding new objects to the key:

```
self.output['connector_type'][0].variable = self.physical_output
self.output_signals['signal_name'][0] = self.output_signal
```

Additional outputs can also be updated when inheriting an existing computeState/computeAction method by calling super().computeAction() and adding the update for new outputs after the call to super().